ICL endeavours to ensure that the information in this document is correct and fairly stated, but does not accept liability for any error or omission.

The development of ICL products and services is continuous and published information may not be up-to-date. Any particular issue of a product may contain part only of the facilities described in this document or may contain facilities not described here. It is important to check the current position with ICL.

Specifications and statements as to performance in this document are ICL estimates intended for general guidance. They may require adjustment in particular circumstances and are therefore not formal offers or undertakings.

Statements in this document are not part of a contract or program product licence save insofar as they are incorporated into a contract or licence by express reference. Issue of this document does not entitle the recipient to access to or use of the products described, and such access or use may be subject to separate contracts or licences.

Technical Publication 4322
© International Computers Limited 1972
First Edition February 1967
Second Edition July 1972
Reprinted June 1977

ICL will be pleased to receive readers' views on the contents and organisation, etc. of this publication. Please write to

The Registry (Readership Survey)
UK Software and Literature Distribution Centre
International Computers Limited
60 Portman Road
Reading
Bersks RG3 1NR

Distributed by
UK Software and Literature Distribution Centre
International Computers Limited
Registered Office: ICL House, Putney, London SW15 1SW
Printed by ICL Printing Services
Works Road, Letchworth, Herts SG6 1JY
Preface

This Reference Manual is intended for use by programmers who have completed a training course in PLAN, and require a day-to-day source of information.

It is primarily a language manual, setting out the rules for the writing of acceptable PLAN programs, together with advice in some instances on efficiency considerations.

Because PLAN is an assembly language, it allows the user access to any facility available to 1900 Series object programs. It is not intended that this manual should describe in full detail all of these facilities; references are made in a number of places to other 1900 Series manuals where more detailed descriptions may be found. Rather, the objective has been to cover those topics which will be of everyday use, in a degree of detail consistent with normal program writing.

The manual is divided into four parts.

Part 1 Basic Information

The Chapter 1 deals with the special characteristics of 1900 Series central processors that are relevant to the writing of programs in PLAN. Chapter 2 gives a basic description of PLAN and how it is used, including the ground rules. Part 1 concludes in Chapter 3 with some details of the more advanced PLAN facilities.

Part 2 Operation Statements and Directives

Chapter 4, 5 and 6 contain descriptions of, respectively, all PLAN operation statements, pseudo-operations and directives. For ease of reference, these descriptions are in alphabetical order within the chapters (or sections for the different facilities in the case of Chapter 5).

Part 3 Recording, Compiling and Testing Programs

Part 3 begins with a description of the conversion of programs from source to object form in Chapter 7, which also includes full details of each PLAN 3, PLAN 2 and PLAN 1 compiler. Chapter 8 deals with the PLAN 4 compilers. The punching of source programs, the compiler listings and the program testing facilities are dealt with in Chapter 9. The creation and maintenance of source programs on magnetic tape are fully described in Chapter 10. The creation and maintenance of source programs on exchangeable disc and on fixed disc are fully described in Chapter 11.

Chapter 12 describes the GEORGE 3 System Macros.

Part 4 Appendices

There are seven appendices, which include information on: the 1900 Series internal and external data codes; octal order codes for PLAN instructions; formats for control areas for peripheral transfers and open and close operations; tables of powers of 2 (decimal together with decimal/octal conversion tables); example compiler listings; overlay programming techniques; and a summary of the PLAN compilers.
PLAN REFERENCE MANUAL PART 1

Basic Information
# Contents

## Preface

## PART 1 BASIC INFORMATION

### Chapter 1 Basic 1900 Information

- **INTRODUCTION**  1
- **WORDS AND SUBDIVISIONS OF WORDS**  1
- **Formats of Data Words**
  - **UNSIGNED BINARY INTEGER**  1
  - **SIGNED BINARY INTEGER**  1
  - **SIGNED BINARY FRACTION**  2
  - **FIXED-POINT NUMBER**  2
  - **FLOATING-POINT NUMBER**  2
  - **CHARACTERS**  3
  - **INDEX WORD**  3
- **Formats of Instruction Words**  4
- **OVERFLOW AND CARRY**  5
- **The Overflow Register**  5
- **The Floating-point Overflow Register**  5
- **The Carry Register**  6
- **MODE**  6
- **Zero Suppression Mode**  6
- **Address Mode**
  - **COMPACT MODE (15-BIT ADDRESS MODE)**  7
  - **EXTENDED DATA MODE (22-BIT ADDRESS MODE)**  7
- **Branch Mode**
  - **DIRECT BRANCH MODE**  7
  - **EXTENDED BRANCH MODE**  7
- **Determination of Mode**  8
- **MODIFICATION**  8
- **NEGATIVE MODIFIERS**  9
- **RESERVED LOCATIONS**  9
- **Locations that are Always Reserved**  9
- **Locations that are Reserved in Subprogramming**  10
- **EXTRACODES**  10
- **MULTIPROGRAMMING AND DUALPROGRAMMING**  10
- **Multiprogramming**  10
- **Dualprogramming**  11

### Chapter 2 Basic Information on PLAN

- **GENERAL FORM OF A PROGRAM**  1
- **USE OF SYMBOLS IN PLAN**  1
Rules for Writing Relative Expressions in the Operand Field of Instruction Statements 16
Relative Expressions in Operand Fields 16
BRANCH INSTRUCTIONS 16
NON-BRANCH INSTRUCTIONS 16
Literal Operands 17
BRANCH INSTRUCTIONS 17
NON-BRANCH INSTRUCTIONS 17
Relative Operands 18
BRANCH INSTRUCTIONS 18
NON-BRANCH INSTRUCTIONS 19
VERSIONS OF PLAN—SUMMARY OF FACILITIES 19
PLAN 1 19
PLAN 2 19
PLAN 3 19
PLAN 4 20
SUMMARY OF OPERATIONS 21-22

Chapter 3 Additional Facilities 1
INTRODUCTION 1
SUBPROGRAMMING 1
PROGRAM SEGMENTATION 1
Segment Intercommunication in PLAN 2, PLAN 3, and PLAN 4 Programs 2
Subroutines 2
Library Subroutines 2
Consolidation of Subroutines 2
OVERLAY PROGRAMMING 2
PROGRAMMING FOR PAGED ENVIRONMENTS 3
Sparse and Dense Programs 3
Shareable Programs and Areas 3

PART 2 OPERATION STATEMENTS AND DIRECTIVES

Chapter 4 Operation Statements 1
INTRODUCTION 1
LAYOUT OF THE CHAPTER 1
ABBREVIATIONS USED IN THE CHAPTER 2
DESCRIPTIONS OF OPERATION STATEMENTS 2
ADN 3
ADNC 4
ADS 5
ADSC 7
ADX 8
ADXC 10
ALLOT 12
ANDN 15
ANDS 16
ANDX 17
<p>| AUTO   | 19 |
| BCC    | 20 |
| BCHX   | 22 |
| BCHX (MACRO-INSTRUCTION) | 24 |
| BCS    | 26 |
| BCT    | 28 |
| BDX    | 30 |
| BDX (MACRO-INSTRUCTION) | 32 |
| BFP    | 34 |
| BNG    | 36 |
| BNZ    | 37 |
| BPZ    | 38 |
| BRN    | 39 |
| BSP    | 40 |
| BTM    | 41 |
| BUX    | 42 |
| BUX (MACRO-INSTRUCTION) | 44 |
| BVC    | 46 |
| BVCI   | 47 |
| BVCR   | 48 |
| BVS    | 49 |
| BVSR   | 50 |
| BXE    | 51 |
| BXGE   | 54 |
| BXL    | 57 |
| BXU    | 60 |
| BZE    | 63 |
| CALL   | 64 |
| CBD    | 66 |
| CDB    | 68 |
| CONT   | 70 |
| CLOSE  | 72 |
| DCH    | 73 |
| DEL    | 75 |
| DELTY  | 77 |
| DEX    | 80 |
| DIS    | 81 |
| DISP   | 83 |
| DISTY  | 84 |
| DLA    | 86 |
| DSA    | 88 |
| DVD    | 90 |
| DVR    | 92 |
| DVS    | 94 |
| ERN    | 96 |
| ERS    | 98 |
| ERX    | 100 |
| EXIT   | 101 |
| FAD    | 103 |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDVD</td>
<td>104</td>
</tr>
<tr>
<td>FIX</td>
<td>106</td>
</tr>
<tr>
<td>FLOAT</td>
<td>107</td>
</tr>
<tr>
<td>FMPY</td>
<td>108</td>
</tr>
<tr>
<td>FSB</td>
<td>109</td>
</tr>
<tr>
<td>FTM</td>
<td>111</td>
</tr>
<tr>
<td>GIVE</td>
<td>112</td>
</tr>
<tr>
<td>LAEZ</td>
<td>119</td>
</tr>
<tr>
<td>LDCH</td>
<td>120</td>
</tr>
<tr>
<td>LDCM</td>
<td>122</td>
</tr>
<tr>
<td>LDCT</td>
<td>124</td>
</tr>
<tr>
<td>LDEX</td>
<td>125</td>
</tr>
<tr>
<td>LDLA</td>
<td>127</td>
</tr>
<tr>
<td>LDN</td>
<td>129</td>
</tr>
<tr>
<td>LDNC</td>
<td>130</td>
</tr>
<tr>
<td>LDPL</td>
<td>131</td>
</tr>
<tr>
<td>LDSA</td>
<td>132</td>
</tr>
<tr>
<td>LDX</td>
<td>134</td>
</tr>
<tr>
<td>LDXC</td>
<td>136</td>
</tr>
<tr>
<td>LEFP</td>
<td>138</td>
</tr>
<tr>
<td>LFP</td>
<td>139</td>
</tr>
<tr>
<td>LFPZ</td>
<td>140</td>
</tr>
<tr>
<td>MODE</td>
<td>141</td>
</tr>
<tr>
<td>MOVE</td>
<td>143</td>
</tr>
<tr>
<td>MPA</td>
<td>145</td>
</tr>
<tr>
<td>MPR</td>
<td>147</td>
</tr>
<tr>
<td>MPY</td>
<td>148</td>
</tr>
<tr>
<td>MVCH</td>
<td>149</td>
</tr>
<tr>
<td>NEFPS</td>
<td>150</td>
</tr>
<tr>
<td>NFPS</td>
<td>151</td>
</tr>
<tr>
<td>NGN</td>
<td>152</td>
</tr>
<tr>
<td>NGNC</td>
<td>153</td>
</tr>
<tr>
<td>NGS</td>
<td>154</td>
</tr>
<tr>
<td>NGSC</td>
<td>156</td>
</tr>
<tr>
<td>NGX</td>
<td>158</td>
</tr>
<tr>
<td>NGXC</td>
<td>160</td>
</tr>
<tr>
<td>NORM</td>
<td>162</td>
</tr>
<tr>
<td>NULL</td>
<td>164</td>
</tr>
<tr>
<td>OBEY</td>
<td>165</td>
</tr>
<tr>
<td>OFF</td>
<td>167</td>
</tr>
<tr>
<td>ON</td>
<td>168</td>
</tr>
<tr>
<td>ORN</td>
<td>169</td>
</tr>
<tr>
<td>ORS</td>
<td>171</td>
</tr>
<tr>
<td>ORX</td>
<td>173</td>
</tr>
<tr>
<td>OVER</td>
<td>174</td>
</tr>
<tr>
<td>PERI</td>
<td>176</td>
</tr>
<tr>
<td>REL</td>
<td>179</td>
</tr>
<tr>
<td>Rew</td>
<td>181</td>
</tr>
<tr>
<td>RFP</td>
<td>182</td>
</tr>
</tbody>
</table>
RRQ 183
SAE 186
SAEZ 187
SBN 188
SBNC 189
SBS 190
SBSC 192
SBX 193
SBXC 195
SCR 196
SEFP 197
SEFPZ 198
SFP 199
SFPZ 200
SLA 201
SLC 203
SLL 205
SMO 207
SRA 208
SRAV 210
SRC 212
SRK 214
STO 216
STOC 218
STOZ 219
SUM 220
SUSAR 221
SUSBY 222
SUSDP 224
SUSIN 227
SUSMA 229
SUSTY 231
SUSWT 233
TEST 234
TXL 235
TXU 237
UNL 239
WTM 240
ZEFPS 241
ZFPS 242

Chapter 5 Pseudo-operations
INTRODUCTION 1
THE INPUT/OUTPUT GENERATOR 1
INPUT 2
INDIS 3
OUT 9
THE VARIABLE LIMIT REGISTER 14
UNOFF 15
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNONC</td>
<td>16</td>
</tr>
<tr>
<td>UNONL</td>
<td>17</td>
</tr>
<tr>
<td>STORAGE DEVICE HOUSEKEEPING</td>
<td>18</td>
</tr>
<tr>
<td>SDBSS</td>
<td>20</td>
</tr>
<tr>
<td>SDBTS</td>
<td>21</td>
</tr>
<tr>
<td>SDBUF</td>
<td>22</td>
</tr>
<tr>
<td>SDCLB</td>
<td>24</td>
</tr>
<tr>
<td>SDCLS</td>
<td>25</td>
</tr>
<tr>
<td>SDCRE</td>
<td>26</td>
</tr>
<tr>
<td>SDDEF</td>
<td>27</td>
</tr>
<tr>
<td>SDDEL</td>
<td>29</td>
</tr>
<tr>
<td>SDEND</td>
<td>30</td>
</tr>
<tr>
<td>SDEXT</td>
<td>31</td>
</tr>
<tr>
<td>SDFAB</td>
<td>32</td>
</tr>
<tr>
<td>SDFES</td>
<td>34</td>
</tr>
<tr>
<td>SDFSS</td>
<td>35</td>
</tr>
<tr>
<td>SDFTS</td>
<td>36</td>
</tr>
<tr>
<td>SDIND</td>
<td>37</td>
</tr>
<tr>
<td>SDLAB</td>
<td>38</td>
</tr>
<tr>
<td>SDRD</td>
<td>40</td>
</tr>
<tr>
<td>SDRDB</td>
<td>41</td>
</tr>
<tr>
<td>SDRDP</td>
<td>43</td>
</tr>
<tr>
<td>SDRRB</td>
<td>44</td>
</tr>
<tr>
<td>SDSUS</td>
<td>46</td>
</tr>
<tr>
<td>SDWR</td>
<td>47</td>
</tr>
<tr>
<td>SDWRB</td>
<td>48</td>
</tr>
<tr>
<td>SDWRI</td>
<td>50</td>
</tr>
<tr>
<td>SDWRS</td>
<td>51</td>
</tr>
<tr>
<td>SDWRU</td>
<td>52</td>
</tr>
<tr>
<td>SDWSS</td>
<td>53</td>
</tr>
<tr>
<td>OVERLAY MACRO-INSTRUCTIONS</td>
<td>55</td>
</tr>
<tr>
<td>ENTER</td>
<td>56</td>
</tr>
<tr>
<td>RECAL</td>
<td>57</td>
</tr>
<tr>
<td>BRING</td>
<td>58</td>
</tr>
<tr>
<td>THE DUMP AND RESTART PACKAGE</td>
<td>59</td>
</tr>
<tr>
<td>SDUMP</td>
<td>60</td>
</tr>
</tbody>
</table>

Chapter 6 Directives

INTRODUCTION

General Description of Directives

STEERING SEGMENTS

Steering Segments for Overlays

DESCRIPTIONS OF PLAN DIRECTIVES

#CMODE

# (COMMENT)

#COMPLETE

#CUE

#DEFINE

#ELASTIC
#END
ENTRY 14
ERRORSEG 15
FINISH 16
HMODE 17
LIBRARY 18
LOWER 19
MACRO 21
MONITOR 24
OMIT 26
ORDER 27
OUST 28
OVERLAY 29
PAGE 30
PERIPHERAL 31
PERMANENT 32
PLOWER 33
PMODE 34
PROGRAM 36
UPPER 41
SET 42
STOP 43
SWITCH 44
UPPER 45

PART 3 RECORDING, COMPILING AND TESTING PROGRAMS

Chapter 7 Compilers
CONVERSION OF SOURCE PROGRAM TO OBJECT PROGRAM 1
Phase 1: Compilation
Phase 2: Consolidation 4
Phase 3: Loading 6
COMPIlers AND FACILITIES 7
PLAN 3 COMPILERS 9
Magnetic Tape and Cassette Tape Usage with PLAN 3 Compilers 10
Object Program Names 10
PLAN 3 COMPILERS WITH BASIC PERIPHERAL INPUT AND OUTPUT 10
The PLAN 3 Compiler #XPLE 10
Form of Source Input 10
Object Program Output 10
Basic Peripheral Usage 11
  INPUT DEVICE 11
  PAPER TAPE PUNCH 11
  CARD PUNCH 11
  LINE PRINTER 11
Example of an Input File for File Recreation
Line Printer Output
THE DISC COSY COMPILERS
ADDITIONAL INFORMATION
Subfile Formats
Magnetic Tape Input Files
Editing Mode of #XPMX and #XPMZ
AMEND-IN-SITU
COPY-AND-AMEND
Input Files for the DISC COSY Compilers
THE LEVEL 0 DIRECTORY SUBFILE
THE LEVEL 1 DIRECTORY SUBFILE
LEVEL 2 DATA SUBFILES

Chapter 12 GEORGE 3 Macros
INTRODUCTION
PRESET STEERING FILES
MACRO PARAMETERS
COMMON PARAMETER TYPES
CHARACTER PERIPHERAL INPUT
LISTING
ERROR
SPECIFICATIONS OF SYSTEM MACROS
PLAN
PLAN 4
PLAN4T
QPLAN
QPLAN4
DACOSYCREATE
DACOSYAMEND
MTCOSYCREATE
MTCOSYAMEND

PART 4 APPENDICES

Appendix 1 ICL 1900 Series Codes
THE 1900 SERIES 64-CHARACTER CARD CODE
THE 8-TRACK (7 DATA BIT) 1900 PAPER TAPE CODE
8-TRACK 1900 PAPER TAPE CODE, USING 7 DATA BITS
8-TRACK 1900 PAPER TAPE CODE, EVEN PARITY
6-BIT, 3 SHIFT INTERNAL MACHINE FORM
THE INTERNAL MACHINE CODE, EXCLUDING SHIFT CHARACTERS
COMPARISON WITH 7-DATA BIT PAPER TAPE CODE
PRINTER CODE

Appendix 2 Tables of Powers of 2
TABLE OF POWERS OF 2 (DECIMAL)
DECIMAL → OCTAL CONVERSION TABLE

XX
Chapter I Basic 1900 Information

INTRODUCTION

A computer program consists of instructions that operate upon data. In the 1900 Series, the store that holds the instructions and the data is made up of words, each word consisting of 24 binary digits or 'bits'. The bits are numbered from 0 (the most significant or 'left-hand') to 23 (the least significant or 'right-hand'); conventionally, Bn is used to refer to bit n of a word.

A program is allocated an area of consecutively numbered words or 'locations' in the computer store. This area will always be a multiple of 64 words, the number of words allocated to each program being made up to the next higher multiple of 64 as part of the compilation process. From the point of view of the programmer, the words in the area are numbered consecutively, starting at 0.

At the time of loading the program, the physical position in which the program is held in the computer store is determined; it may vary in some cases during the time that the program is running. Such positioning and repositioning is allowed for automatically by the hardware of the processor, and need not concern the programmer at all.

A program will also involve the use of peripheral devices (such as card readers, printers, magnetic tape units). Each of these can be addressed by the programmer by means of a device code and a number, all units of one type being numbered separately. Just as the hardware adjusts the programmer's addressing of the core store to the actual physical position, so the programmer's numbering of peripherals is matched to the actual situation at the time of running the program. In this case, however, the matching is carried out by the Executive rather than hardware. Again, the programmer need not concern himself with this process, but simply use his own numbering of units.

Freedom from absolute addressing is useful in a processor that runs only one program at a time, simplifying, for instance, the interchange of programs between installations. It is an essential requirement for multiprogrammed operation, where each program must be free to run in combination with others without prior collision at the time of writing.

When a program is presented for loading, Executive allocates the peripheral units and the area in core store to meet the program's initial requirements. (If the initial needs cannot be met the program is rejected.) This allocation may be altered subsequently, either at the request of the program or as other circumstances change.

WORDS AND SUBDIVISIONS OF WORDS

This section contains descriptions of the formats of all the types of words that are associated with a program. The section is in two parts, one dealing with data words and the other with instruction words.

The 24 binary digits present in each word are referred to, from left to right, as B0, B1, B2, etc., up to B23.

Formats of Data Words

UNSIGNED BINARY INTEGER

The word represents a 24-bit unsigned binary integer, in which B23 is the least significant digit position.

SIGNED BINARY INTEGER

The integer is contained in B1 to B23, the sign being in B0. If B0 is zero, the integer is positive (or zero), and if B0 is one, the integer is negative. If the integer is negative, it is held in two's complement form where the radix is two. The integer lies in the range \(-2^{23}\) to \(+2^{23} - 1\).
Some examples of signed binary integers are given in the following table:

<table>
<thead>
<tr>
<th>Integer</th>
<th>Binary Representation in a 24-bit Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>0 00000 000000 000000 000001</td>
</tr>
<tr>
<td>+13</td>
<td>0 00000 000000 000000 001101</td>
</tr>
<tr>
<td>-1</td>
<td>1 11111 111111 111111 111111</td>
</tr>
<tr>
<td>-13</td>
<td>1 11111 111111 111111 110011</td>
</tr>
<tr>
<td>+128 = 2^7</td>
<td>0 00000 000000 000010 000000</td>
</tr>
<tr>
<td>-128</td>
<td>1 11111 111111 111110 000000</td>
</tr>
<tr>
<td>+4, 194, 304 = 2^{22}</td>
<td>0 10000 000000 000000 000000</td>
</tr>
</tbody>
</table>

**SIGNED BINARY FRACTION**

The fraction is contained in B1 to B23, the sign being indicated in B0. If B0 is zero, the fraction is positive (or zero), and if B0 is one, the fraction is negative. The binary point may be considered to be to the right of B0 and to the left of B1. If the fraction is negative, it is held in two’s complement form where the radix is two. The fraction lies in the range -1.0 to +1.0 - 2^{-23}.

Some examples of signed binary fractions are given in the following table:

<table>
<thead>
<tr>
<th>Fraction</th>
<th>Binary Representation in a 24-bit Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1/2</td>
<td>0 10000 000000 000000 000000</td>
</tr>
<tr>
<td>+1/8 = 2^{-3}</td>
<td>0 00100 000000 000000 000000</td>
</tr>
<tr>
<td>-1/8</td>
<td>1 11100 000000 000000 000000</td>
</tr>
<tr>
<td>+13 x 2^{-23}</td>
<td>0 00000 000000 000000 001101</td>
</tr>
<tr>
<td>-13 x 2^{-23}</td>
<td>1 11111 111111 111111 110011</td>
</tr>
<tr>
<td>+7/8</td>
<td>0 11000 000000 000000 000000</td>
</tr>
<tr>
<td>-3/8</td>
<td>1 01000 000000 000000 000000</td>
</tr>
</tbody>
</table>

**FIXED-POINT NUMBER**

A fixed-point number may be contained in one or more words that are usually adjacent. Part of the number is regarded as an integer and part as a fraction. If the number is held in one word, the sign is indicated in B0. If the number is held in two or more words, the sign is indicated in B0 of the word containing the most significant part of the number, and B0 of each of the remaining words is zero.

A common use of the fixed-point number is the double-length mid-point number, in which the integral part is held in the lower-numbered word and the fractional part in the other word. The binary point is then considered to be between the two words.

\[
\begin{array}{c|c|c|c}
\hline
S & 0 & 23 & 1 & 23 \\
\hline
1 & & & & \\
\hline
\end{array}
\]

S is sign bit, 0 is B0 of least significant word

*Double-length Fixed-point Number*

**FLOATING-POINT NUMBER**

A floating-point number is contained in two adjacent words, the bits in the higher-numbered word being referred to as B24 to B47.

Any floating-point number may be represented by

\[a \cdot r^b\]

where \(a\) is the fixed-point part (the argument)
For 1900 computers, \( r = 2 \) and \( a \) is a signed binary fraction that extends over the range of \( B0 \) to \( B23 \) and \( B25 \) to \( B38 \); \( B39 \) to \( B47 \) contain an unsigned binary number that equals \( b + 256 \). \( B24 \) is used to indicate floating-point overflow conditions (see page 5).

The standard range for the argument \( a \) is:
\[
\frac{1}{2} < a < 1 \quad \text{or} \quad -1 < a < -\frac{1}{2}
\]

During the course of arithmetic operations on floating-point numbers, the argument may go out of range. The process of shifting the argument into standard range and adjusting the exponent is termed normalization; functions are provided to do this for both single- and double-length floating-point numbers.

The exponent is not recorded as \( b \), but as \( b + 256 \). Hence the exponent part of a floating-point number will never be negative, and standard floating-point zero \((0 \times 2^{-256})\) has both argument and exponent as zero. Consequently, floating-point zero can be detected by the same 'branch on zero' instructions as fixed-point zero.

\[\begin{array}{c|c|c|c}
S & 0 & x_b \\
1 & 23 & 1 & 14 & 9
\end{array}\]

\( S \) is sign bit, 0 is \( B0 \) of least significant word, \( x_b \) contains exponent plus 256

**Floating-point Number**

**CHARACTERS**

Six consecutive bits can represent, according to a defined code, any of 64 characters, including letters or digits (see Appendix 1). One word can thus contain four characters, with

- character 0 in \( B0 \) to \( B5 \)
- character 1 in \( B6 \) to \( B11 \)
- character 2 in \( B12 \) to \( B17 \)
- character 3 in \( B18 \) to \( B23 \).

These four character positions are usually referred to, from left to right, as \( n0 \), \( n1 \), \( n2 \) and \( n3 \).

\[\begin{array}{c|c|c|c}
n0 & n1 & n2 & n3 \\
6 & 6 & 6 & 6
\end{array}\]

**Characters**

**INDEX WORD**

There are two types of index word: counter-modifiers and character counter-modifiers.

1. In a counter-modifier, \( B0 \) to \( B8 \) contain the counter and \( B9 \) to \( B23 \) contain the modifier. (When operating in extended mode, two words are used. \( B9 \) to \( B23 \) of one contain the counter, and \( B2 \) to \( B23 \) of the other contain the modifier.)

2. In a character counter-modifier, \( B0 \) and \( B1 \) contain the character modifier \((0, 1, 2 \text{ or } 3)\), \( B2 \) to \( B8 \) contain the counter and \( B9 \) to \( B23 \) contain the word modifier. (When operating in extended data mode, two words are used. \( B9 \) to \( B23 \) of one contain the counter, and, of the other, \( B0 \) and \( B1 \) contain the character modifier and \( B2 \) to \( B23 \) contain the word modifier.)
\[
\begin{array}{|c|c|}
\hline
x_c & x_d \\
\hline
9 & 15 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
x_k & x_l & x_m \\
\hline
2 & 7 & 15 \\
\hline
\end{array}
\]

- \(x_c\) contains counter,
- \(x_m\) contains modifier

Counter - modifier

- \(x_k\) contains character modifier,
- \(x_l\) contains counter
- \(x_m\) contains word modifier

Character Counter - modifier

**Formats of Instruction Words**

The formats of instruction words are given in the following table:

<table>
<thead>
<tr>
<th>TYPE OF INSTRUCTION</th>
<th>X-FIELD</th>
<th>F-FIELD</th>
<th>M-FIELD</th>
<th>N-FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRANCH INSTRUCTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 DIRECT BRANCH MODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 EXTENDED BRANCH MODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NON-BRANCH INSTRUCTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 NORMAL INSTRUCTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 SHIFT INSTRUCTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B0 to B2 (Sometimes refers to an accumulator)</td>
<td>B3 to B8 (The machine code number of the branch function)*</td>
<td>-</td>
<td>B9 to B23 (The address of the destination)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R-FIELD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N-FIELD</td>
</tr>
<tr>
<td></td>
<td>B9 (Determines whether relative or replaced branch)</td>
<td>-</td>
<td>B10 to B23 (The relative address of the destination or the address of the replacer)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B12 to B23 (The operand information)</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N1-FIELD</td>
<td>N2-FIELD</td>
<td>B12 and B13 (Determines what type shift is performed)</td>
<td>B14 to B23 (The number of places of shift)</td>
</tr>
</tbody>
</table>

* The machine-code numbers for all branch functions are even, so they can be represented in six bits by ignoring a seventh, least significant bit that adds one for odd numbers.
OVERFLOW AND CARRY

The Overflow Register

The Overflow of V register is a special one-bit register associated with, but outside, the program. Overflow is set (V = 1) when the result of carrying out an instruction in single-length working exceeds 23 bits plus sign (the relevant limits are indicated on page 1). Once set, V remains set until explicitly cleared.

The instructions that may set V are:

LDX  ADX  NGX  SBX
STO  ADS  NGS  SBS
MPY  MPR  MPA
DVD  DVR  DVS
ADN  SBN
BVCI
SLA
FIX  SFP  SFPZ  NORM
CDB
EXIT
BFP  NEFPS  NFPS  SEFP  SEFPZ

Overflow is cleared (V = 0), if it was set, by the following instructions:

CALL  BVSR  BVCR  BVCI  NORM

The following instruction clears V, if it was set, unless the number of places of shift specified (N_f, plus the modifier, if any) is zero:

SRAV

The action of the following branch instructions depends upon the state of V:

BVS  BVSR  BVCR  BVCI

The Floating-point Overflow Register

If, during any operation in the floating-point accumulator, A, the exponent goes out of range, B24 of the double-length word A is set to 1. Floating-point Overflow, FOVR, is then said to be set.

The state of FOVR can be tested directly only on the 1902A, 1903A and processors with extended data mode facilities, by the BFP instruction. However, V will be set if a FIX, SFP or SFPZ instruction is given while FOVR is set, or if a BFP instruction with X = 0, 1, 2 or 3 is given while FOVR is set.
Floating-point overflow may be set (FOVR = 1) by the following instructions:

- FAD
- FSB
- FMPY
- FDVD
- LEFP
- LFP
- NEFPS
- NFPS

Floating-point overflow is cleared (FOVR = 0) by the following instructions:

- LFPZ
- SEFPZ
- SFPZ
- ZEPS
- ZFPS

and may be cleared by LFP.

The Carry Register

The Carry or C register is a special one-bit register associated with, but outside, the program. Carry is set (C = 1) when the result of carrying out an instruction in multi-length working exceeds the capacity.

For the following instructions, the specified operations are first performed on the 24-bit signed operands and C is set if, in the result, B0 = 1:

- LDXC
- ADXC
- STOC
- ADSC
- ADNC

For the following instructions, the specified operations are first performed on the operands treated as 24-bit positive numbers and C is set if, in the result, B(-1) = 1:

- NGXC
- SBXC
- NGSC
- SBSC
- NGNC
- SBNC

On completion of any of the above eleven instructions B0 is always left as zero.

The following instructions may also set C, as explained in their entries in Chapter 4:

- TXU
- TXL
- CDB

Carry is cleared (C = 0) by any instruction (except OBEY, SMO and NULL, and sometimes TXU and TXL), unless that instruction would result in C being set.

The action of the following branch instructions depends upon the state of C:

- BCS
- BCC

MODE

Certain details of the internal operation of the central processor are determined by mode settings. These internal operating modes, which are determined separately for each program member, fall into three categories: zero suppression mode, address mode and branch mode.

Zero Suppression Mode

Zero suppression mode is relevant when a CBD instruction is used to convert a binary number to a string of decimal characters. The mode may be set to 0 or 1 by a MODE instruction. If the mode is set to 1 then, when a CBD instruction is given, zero suppression occurs, i.e. leading zeros are replaced by space characters. If the mode is set to 0 there is no zero suppression. The mode may also be set by a CALL or EXIT instruction. (The CBD, MODE, CALL or EXIT instructions are fully described in Chapter 4.)

Address Mode

The 12-bit N-field of a non-branch instruction permits the direct addressing of any location up to location 4095 of the program. If a location higher than 4095 is to be addressed, then modification must be used to extend the effective addressing-range of the instruction. Modification is described in a later section.

Some processors in the 1900 series cannot have a core store greater in size than 32,768 words, and with such processors a 15-bit modifier is sufficient to enable an instruction to address any location in the store. The larger central processors may contain programs that need to address data locations above 32,767, and therefore require a modifier of more than 15 bits. A choice of address modes is provided so that the larger processors are capable of running programs written to run also in the smaller machines.
COMPACT MODE (15-BIT ADDRESS MODE)

In compact mode, all modified addresses in non-branch instructions are truncated to 15 bits; hence a synonym for compact mode is 15-bit address mode, and the standard abbreviation for this mode is 15AM.

A program must be able to operate in compact mode if it is to run successfully in central processors with a maximum storage capability of 32,768 words or less.

EXTENDED DATA MODE (22-BIT ADDRESS MODE)

Extended data mode is available only on processors which are capable of driving a store greater in size than 32,768 words and on which a suitable Executive is provided. In extended data mode all modified addresses are truncated to 22 bits; hence a synonym for extended data mode is 22-bit address mode, and the standard abbreviation for this mode is 22AM.

Certain instructions operate differently in extended data mode. The differences are described under the individual instructions affected, in Chapter 4. There are certain additional instructions (BCT, MVCH, SMO and BFP) which are available only on 1902A and 1903A processors and on processors with extended data mode facilities; these instructions may be used on the latter group of processors in compact mode or in extended data mode, and on the 1902A and 1903A in compact mode only.

Branch Mode

The 15-bit N-field of a branch instruction permits a program to branch directly to any program location up to 32,767. With the larger central processors this 15-bit limitation would restrict program instructions to the first 32,768 words of the program area, were means not provided to permit branching to a wider range of program addresses, thus enabling full advantage to be taken of the greater storage capabilities of these machines. The supplementary modification facility (see the description of the SMO instruction in Chapter 4) is one method of achieving this; another method, in central processors with a storage capability greater than 32,768 words, is the provision of different modes of operation for branch instructions. A choice of branch modes is provided so that the larger processors are capable of running programs written to run also in the smaller machines.

DIRECT BRANCH MODE

When a program member is operating in direct branch mode, the destination address of a branch instruction (i.e. the location to which control will be transferred if the branch takes place) is specified by the 15 bits of the N-field, modified if applicable by a supplementary modifier. The modification by a supplementary modifier applies only if

1. the processor in which the program is running has extended data mode capabilities, and
2. the branch instruction is immediately preceded by a SMO instruction, or by an OBEY instruction of which a SMO instruction is the ultimate operand.

Note that if supplementary modification is employed in direct branch mode, the modified destination address will extend over 15 or 22 bits, according to whether the program member is operating in compact mode or in extended data mode.

The standard abbreviation for direct branch mode is DBM.

EXTENDED BRANCH MODE

Extended branch mode introduces two further types of branches: relative branches and replaced branches. Bit 9 of the instruction is used to indicate the type of branch, thus leaving a 14-bit N-field. If B9 is zero then the instruction is a relative branch; if B9 is one, then the instruction is a replaced branch.

Relative branches branch to a destination address which is determined relative to the location holding the branch instruction itself. The 14 bits of the N-field are first extended to 22 bits by propagating the value of the most significant bit; the resulting 22-bit number is then added to the address of the location containing the branch instruction, and the sum is truncated to 22 bits to give the destination address. The N-field is thus treated as containing a signed relative address, permitting forward or backward branching; the extension to 22 bits ensures correct subtraction effects when N is negative.
With *replaced branches*, the N-field contains the address of a location which contains the destination address in its least significant 22 bits. The location which contains the destination address is called the ‘replacer’, since in effect its contents ‘replace’ the N-field of the instruction; the technique is alternatively known as indirect addressing.

**Supplementary modification**: Both relative and replaced branches may be subjected to supplementary modification. In each case, the supplementary modifier is added to the destination address determined as described above, and the result is truncated to 22 bits to give the modified destination address. If the program member is working in compact mode, the supplementary modifier is first extended from 15 bits to 22 bits by propagating the value of the most significant bit, so as to preserve any negative modification effect.

The standard abbreviation for extended branch mode is EBM.

**Determination of Mode**

The initial setting of zero suppression mode when a program is loaded is indeterminate. It may subsequently be set to 0 or 1 by a MODE instruction, and will be set to 0 by a CBD instruction on the conversion of the first non-zero character. The setting applies only to the program member issuing the instruction. The CALL instruction will leave the mode set to 0 if it was initially 0, otherwise the setting is indeterminate. The EXIT instruction will restore the mode as bit 8 of the link accumulator if the program is in 15AM and DBM, otherwise as bit 1 of the link accumulator. These instructions are described in Chapter 4.

The initial settings of address mode and branch mode when a program is loaded are determined by the contents of, or the absence of, the supplementary request block. The supplementary request block is described in detail in the Central Processors manual, Chapter 7. If no supplementary request block is present, the program will operate initially in compact mode and direct branch mode. If a supplementary request block is present, the initial address mode and branch mode of member 0 are determined by the values of bits 23 and 21 of the mode word (word 1 of the supplementary request block). The initial address mode and branch mode of members other than member 0 are determined by the mode settings of the activating member at the time that they are first activated by an AUTO instruction. A supplementary request block will not be accepted by central processors that have not got extended data mode facilities; its presence will cause rejection of the program on such machines. The address mode and branch mode of individual program members may be changed subsequently by a GIVE instruction with N(M)=9, and may be interrogated by a GIVE instruction with N(M)=8, as described in Chapter 4.

PLAN compilers do not produce supplementary request blocks. PLAN 4 provides means of specifying in which address mode the program is required to operate initially and in which branch mode it is to operate, and the compilers pass this information on to the loader. The consolidation process permits only one branch mode to be used throughout a program. The loader program itself is loaded in compact mode and direct branch mode but before overwriting itself (see Chapter 7, page 6) it switches the address mode and the branch mode to those specified by the programme. For the method of specifying the initial address mode and the required branch mode, see under the #PMODE directive in Chapter 6. With PLAN 3, PLAN 2 and PLAN 1 compilers no means is provided of specifying initial operating modes, so all programs compiled by these compilers will operate initially in compact mode and direct branch mode.

Since PLAN compilers do not cater for programs which require to switch their branch mode, any such programs have to be written in machine code.

When a program is dumped, a supplementary request block is produced if the operating modes of member 0 at that time are any combination other than compact mode and direct branch mode.

**MODIFICATION**

Modified instructions will have one of the values 1, 2 or 3 in the M-field of the 1900 instruction word. Unmodified instructions have zero in the M-field. If the M-field is 1, 2 or 3, the effect of the N-field is determined by the information contained therein modified by the current information contained in X1, X2 or X3 respectively. If the M-field is zero, when the instruction is performed the effect of the N-field is determined solely by the information contained therein.

Modification takes place in the following manner. When the computer obey a modified non-branch instruction, it first adds the 15 least significant bits (if in compact mode) or the 22 least significant bits (if in extended data mode) of the register specified in the M-field to the 12 bits of the N-field. The sum is truncated to 15 bits (compact mode) or 22 bits (extended data mode), and then operates as the effective operand of the instruction.
With shift instructions, if the modified operand exceeds 10 significant bits the type of shift performed may be altered.

In the case of instructions using a character counter-modifier, the two most significant bits of the register specified in the M-field are also used, to specify the character position within the word addressed by the modified operand.

Additionally, both branch instructions and modified or unmodified non-branch instructions may be subjected to supplementary modification by means of a preceding SMO instruction. The effect of this instruction is described in Chapter 4.

NEGATIVE MODIFIERS

As implied by the above definitions of modification, it is possible to modify by a small negative integer. In compact mode this negative integer can be regarded as the difference between the number and $2^{15}$, e.g. -1 can be held as $00000000$ so that the rest of the word can be used as a counter and/or character modifier. In extended data mode the negative integer can be regarded as the difference between the number and $2^{22}$, e.g. -1 can be held as $11111111$ so that B0 and B1 can be used to address a specified character. If it is not intended to use the counter or address a character, the negative integer may be held in the usual form, i.e. as the difference between the number and $2^{24}$.

The 15- and 22-bit number, which after modification constitutes the operand field entry for the instruction, is interpreted as an unsigned binary integer so that $n \cdot (-m)$ must be $>0$, where $n$ and $m$ are respectively the contents of the N-field and the modifier portion of the specified accumulator.

RESERVED LOCATIONS

Certain locations at the low-address end of the area in core store allocated to an object program are reserved for special purposes. Their use by the object program is therefore restricted or prohibited entirely, as detailed in the descriptions that follow.

Locations that are Always Reserved

WORDS 0 TO 7 These are accumulator registers, often referred to as X0 to X7. The accumulators X1, X2 and X3 are the modifier registers.

These words may be used by object programs as operands in both the X and N fields. In cases where the X field of an instruction refers to two adjacent accumulators X and X+1, if 7 is quoted as X then 0 is taken as X+1. In cases where the N field of an instruction is interpreted as a double length operand (as for example in floating-point orders), N(M) must not address location 7.

WORD 8 This location is used by Executive and the central processor hardware for control purposes. It must never be used by the object program.

WORD 9 This location is used by Executive to provide reply information to the object program after an ALLOT instruction; on certain processors it is also used by the hardware.

Object programs may use word 9 in accordance with the specification for ALLOT (see Chapter 4).

WORDS 10 AND 11 These locations are reserved for communication between object programs and some software packages (e.g. Magnetic Tape Housekeeping). Object programs may use these words only in accordance with the relevant software specifications.

WORDS 12 AND 13 These locations are used by Executive and must not be used by object programs.

WORDS 14 AND 15 These locations are reserved for communications between object programs and some software packages. Object programs may use these words only in accordance with the relevant software specifications.

WORDS 16 TO 29 These locations are reserved for use by various compilation systems. Object programs may use these words only in the manner defined by the compiler specifications. For instance, words 20 to 29 are used in PLAN as start points when programs are set into an active state.
WORD 30
This location contains the 24 switch bits. The switches are numbered from 0 to 23, switch 0 being represented by B0 of the word, and so on. The 'ON' state of a switch is represented by the appropriate bit being set to 1. Switches may be set or unset by operators, by means of the ON, OFF or ALTER console directives. Object programs may examine or alter the state of any bit in word 30.

WORD 31
This location is reserved for use by various compilation systems. Object programs may use this word only in the manner defined by the compiler specifications.

WORDS 32 TO 44
These locations are always reserved, and must not be used by object programs. For programs without subprograms or members, words 35 to 44 are used in connection with the General Purpose Loader. Otherwise, these words are reserved for a different use (see below).

Locations that are Reserved in Subprogramming
The following locations are reserved for use by Executive when a program consists of two or more subprograms or members. When these circumstances exist, the relevant words may not be used by object programs. Words 0 to 15 of each subprogram or member are stored in these locations.

WORDS 32 TO 63
These locations are reserved if the program consists of two or more subprograms or members.

WORDS 64 TO 79
These locations are reserved if the program consists of three or more subprograms or members.

WORDS 80 TO 95
These locations are reserved if the program consists of four subprograms or members.

NOTE: The references above to the number of subprograms and members assume that they are consecutively numbered, starting from one. Should this not be so, the reserved area must be calculated on the basis of n subprograms or members, where n is the highest number used to refer to a subprogram or member.

EXTRACODES
On some processors some instructions are performed by machine hardware, while on other processors the same instructions are performed by Executive routines known as Extracodes. A programmer need never consider whether instructions are performed by extracode or by hardware, except from the point of view of timing, when it is sometimes pertinent to avoid using extracodes, if possible.

MULTIPROGRAMMING AND DUALPROGRAMMING

Multiprogramming
The multiprogramming facilities permit optimum use of a 1900 system by enabling a number of main programs to share the time of the central processor. These facilities are available only on 1904 to 1907 central processors, and on all E, F and A models of the 1900 series with the exception of the 1901A, provided that appropriate versions of Executive are used.

The programs are stored within the computer at the same time; when any program is forced to wait for a peripheral device, an alternative program that is free to proceed can use the central processor time that would otherwise be wasted.

Programs are allocated priority numbers so that the appropriate programs have first call upon central processor time when they are free to use it.

When a program is loaded into store and the multiprogramming facilities are in use, the absolute addresses of the first and last locations occupied by the program are held by Executive as the program's 'Datum' and 'Limit'. Before any instruction is obeyed, central processor hardware adds the Datum (and modifier, if any) to the address and checks that the absolute address thus formed does not lie outside the program's reservation (i.e., below Datum or above Limit). If this check fails, the instruction is not carried out and there is an automatic jump into Executive, which suspends the program.
According to the central processor involved, there may be up to 16 programs, each consisting of sub-programs or members if required, using the multiprogramming facilities.

**Dualprogramming**

Dualprogramming facilities are available with 1902 and 1903 central processors with a core store of at least 16,384 words. These facilities are similar to those provided by multiprogramming, except that they apply to the running of only two programs (each of which may consist of subprograms or members if required). Executive checks the effective absolute address against the Datum but not against the Limit.
Chapter 2 Basic Information on PLAN

GENERAL FORM OF A PROGRAM

Using the PLAN language, the work to be done by a program can be described in an unambiguous and logical manner. The language uses symbolic names rather than numbers to describe both program operations and data, which makes programs easier both to write and to read.

There are two main phases in the production of a program using the PLAN language. The first phase involves describing the program in terms of statements written on PLAN coding sheets. These statements are then transcribed on to a computer input medium such as punched cards or paper tape, and the second phase involves using a PLAN compiler on a 1900 series computer to transform the program into the machine coding in which it will exist in the program store when it is being used. The program in PLAN is the source program; the program in machine coding is the object program.

A PLAN program consists of a sequence of statements which have positional significance in relation to each other.

There are three types of statement:
1. Operation (or instruction) statements.
2. Data statements.

OPERATION STATEMENTS - Depending upon the nature of the operation involved, each operation statement will give rise to one or more machine-code instructions. Each machine-code instruction will cause a specific operation to be performed.

DATA STATEMENTS are of two types. Variable data statements request a specified number of storage locations to be reserved for data; constant data statements set into a location a constant value for use by instruction statements.

DIRECTIVE STATEMENTS give a compiler information about the organization of an object program. (They do not result in any machine-code instructions in the object program.)

PLAN statements may occur in any sequence in a program subject to the following restrictions:
1. Every program or segment must start with a #PROGRAM directive which specifies the program or segment name.
2. Every program or segment must end with a #END directive (in PLAN 3 the last segment of a program must end with a #FINISH directive following the #END directive).

USE OF SYMBOLS IN PLAN

The basic element of any programming language is the symbol, which is some combination of characters representing a name. In PLAN specifically, symbols are combinations of alphabetic and numeric characters of which the first character must be alphabetic. Characters such as comma or point are not allowed. Names thus formed usually represent one of the following types:
1. Operation codes.
2. Pseudo-operation codes.
3. Permanent macros.
4. Directives.
5. Store locations.
6. Peripheral units.
The process of compilation consists, essentially, of interpreting these symbols and, where applicable, converting them to their machine language equivalents. Each symbolic operation, pseudo-operation or macro code corresponds to a machine function code (or codes) and each directive represents a special action to be taken by the compiler. The actual names used for these purposes have been assigned and are a permanent part of the PLAN language. Location names, on the other hand, are assigned by the programmer. The actual location represented by a name depends on where and how the name is used in the source program.

STORE LAYOUT
The core store required by a program may comprise three areas:

1 THE LOWER DATA STORE
   (a) Variable Area
   (b) Literals
   (c) Preset Area

2 THE PROGRAM OPERATION STORE
   (Program instructions. Also data if necessary)

3 THE UPPER DATA STORE
   (a) Preset Area
   (b) Variable Area

The lower data store must be completely within the first 4,096 locations used by the program (known as lower memory) whereas the program operation and upper data stores may fall beyond this range (i.e. in upper memory). A distinction is drawn between lower and upper memory because operation statements which refer to data locations have only 12 bits with which to do so. Operation statements refer indirectly to data locations outside the first 4,096 by means of modification. To refer to program locations a type of operation statement known as a branch instruction is used which has a 15-bit operand, making it possible to address directly locations up to 32,767 in direct branch mode (see Chapter 1).

As the compiler encounters symbolic locations in the source program it assigns a numeric value to each one. This involves determining the area into which each one should be placed, and its relative position within that area.

Features of the lower data, program operation and upper data stores are described in greater detail below; in terms of PLAN 2, PLAN 3 and PLAN 4 only. A note on PLAN 1 is given on the next page.

The Lower Data Store
Since data locations may only be referred to directly if they are in lower memory, this area of store will hold the data most frequently used by a program. The PLAN 2, 3 and 4 compilers will consider any symbol which

1 is encountered in the operand field of a non-branch instruction, and

2 has not previously appeared under some defining directive.

When each such symbol is first encountered by the compiler, space is allocated in lower memory. Two consecutive locations will be assigned if the symbol is in the operand of a double-length macro-instruction (the symbol is given the value of the first location); otherwise one location will be assigned. There is, therefore, no need to specify explicitly, in a source program, data locations to be held in the lower data store. If it is required to control the sequence in which data is held in lower memory, however, or to allocate areas of more than two words, it is necessary to specify symbolic locations under the control of the #LOWER directive.

The Program Operation Store
As the compiler reads in source program instructions, it assigns consecutive locations in the program operation store to them. Any symbol appearing in the label field of an instruction (or of a data statement occurring in the body of a program) will be given a value equal to its location, or first location if it occupies more than one. When the symbol is encountered in an operand field it is replaced by this value. A symbol may appear in an operand field without having previously occurred in a label field. A symbol
will be recognized as referring to a program location, therefore, if
1. it appears in a label field in the program, or
2. it appears in the operand field of a branch instruction.

Since references to program locations most frequently occur in the operands of branch instructions, and since the structure of these instructions allows for 15-bit addresses, the program operation store can be in upper memory.

The Upper Data Store

The upper data store may be held entirely in upper memory. The PLAN 2, 3 and 4 compilers will assign areas of upper data to symbolic locations specified under a #UPPER directive. Data in upper memory may only be addressed indirectly, by modification, and the upper data store should, therefore, be used for information which is naturally addressed indirectly (e.g., tables and arrays), or for data only infrequently referred to.

Note on PLAN 1 Store Layout

PLAN 1 programs occupy lower memory only. No distinction is made, therefore, between lower and upper data stores, and #LOWER and #UPPER directives in the program do not affect compilation. The concept of a program operation store also has no meaning in PLAN 1, and #PROGRAM directives do not affect compilation (apart from the first, giving the program or segment name). All these directives may be used as normal in a PLAN 1 program and will be listed; if the program is ever to be compiled using a PLAN 2, 3 or 4 compiler they must be used and in accordance with the rules of those languages.

In PLAN 1, storage is allocated in two areas, preset and non-preset. The preset area contains instructions and preset data statements, in the order in which these occur in the source program. Variable data statements are allocated non-preset storage. In the object program the preset area follows after the non-preset.

In PLAN 1, symbols cannot be defined implicitly by their appearance in the operand field of an instruction. Symbols can only be defined in one of the following ways:
1. By occurrence in the label field of an instruction or preset data statement.
2. By occurrence in a variable data statement.
3. By a #DEFINE directive.

If the following sequence is adhered to, the store layout of a program written in PLAN 1 will be the same as it would be if the program were compiled using a PLAN 2, PLAN 3 or PLAN 4 compiler.

Directives:
- Directive #LOWER specifying Variable Data Statements.
- Directive #LOWER specifying Constant Data Statements.
- Directive #PROGRAM specifying Program Statements.

THE PLAN CODING SHEET

PLAN programs are conveniently written on standard PLAN coding sheets (ICL Form 1/1916), one of which is shown on page 4. One statement is written per line, which corresponds to one punched card or one line of characters on paper tape.

Program documentation presented for punching into cards or paper tape must be legible and unambiguous. In particular, certain symbols that are similar to other symbols should be written according to agreed conventions that are clearly understood by both the programmers and the punch operator.

The following PLAN program writing conventions are recommended:

- The digit 'one': 1
- The digit 'zero': 0
- The digit 'two': 2
- The letter: I
- The letter: O
- The letter: D

It is useful to be able to give a positive indication of a space character where this is significant.
### ICL 1900 Series PLAN coding sheet

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Acc</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>12</td>
<td>15-16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>24-26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>32-40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40</td>
<td>44-48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>52</td>
<td>56-60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60</td>
<td>64-68</td>
</tr>
<tr>
<td></td>
<td></td>
<td>72</td>
<td>78-80</td>
</tr>
</tbody>
</table>

---

Chapter 2

© International Computers Limited 1966 Printed in Great Britain
e.g. PROGRAM/TAPE indicating one space character between the word PROGRAM and the word TAPE. Particular care should also be paid to the writing of V and U, 5 and S, 6 and G, 7, 4 and 9.

Each line of the coding sheet is divided into six fields, equivalent to the possible fields in any statement, as follows:

- Character positions 1 to 5, Label field for operation statement
- Character positions 1 to 11, Label field for constant data statement
- Character positions 7 to 11, Operation field
- Character positions 13 to 14, Accumulator field
- Character positions 16 to 72, Operand field
- Character positions 73 to 75, Program identity field
- Character positions 76 to 80, Sequence field.

The uses of these different fields are dealt with below.

Each field except the operand must be written left-justified. In the operand field, blank columns or spaces are ignored unless they constitute part of an 'H' statement.

This is not true in PLAN 1, where a blank column or space in the operand field will terminate a source line statement unless the space constitutes part of an 'H' statement.

The Label Field

OPERATION STATEMENTS

Any instruction may be given a symbolic name or label, so that it can be referred to from other parts of the program (for instance, in a branch instruction). A label is written in the label field of the instruction to which it refers, and must be left-justified. It consists of up to five alphanumeric characters, of which the first must be alphabetic. Labels in a segment must be unique; they have no significance if referred to in any other segment.

DATA STATEMENTS

Preset data statements may be labelled in the same way as operation statements (see above) except that the label may be up to eleven alphanumeric characters in length. If a name is written in this field, then it pertains to the first item of data on the line.

DIRECTIVES

The names of directives are written in the label field. The field always starts with the character # in column 1, followed by a name indicating the type of directive.

The Operation Field

OPERATION STATEMENTS

The operation field contains the name of any PLAN instruction, the name of a user-defined macro-instruction, or the name of a pseudo-operation. All entries must be left-justified. (It is permissible, though not recommended, to replace PLAN instruction names by the three-digit machine codes, written left-justified.)

DATA STATEMENTS

If a long label (more than five characters) is used with a constant data statement it may extend into the operation field (except in PLAN 1).

DIRECTIVES

Some directive names extend from the label field into the operation field.
The Accumulator Field

OPERATION STATEMENTS

The accumulator field may contain either
1. a digit in the range 0 to 7 referring to an accumulator register,
2. two digits each in the range 0 to 7, referring to adjacent accumulator registers,
3. a decimal number in the range 0 to 15, referring to a file number (e.g. SD macro-instructions),
4. a digit in the range 0 to 7 referring to a peripheral unit number (e.g. PERI instructions),
5. a digit in the range 0 to 7 used to qualify the function specified in the operation field, (e.g. floating-point instructions).

Digits in the accumulator field are written left-justified; the field may be blank.

DATA STATEMENTS

The accumulator field is never used.

DIRECTIVES

The accumulator field is never used.

The Operand Field

OPERATION STATEMENTS

The operand field is of variable length and usually indicates the data involved in the operation. It may contain any operand specified in the section in this chapter dealing with types of operand in operation statements (see page 13 et. seq.)

The operand results in the generation of the address portion (i.e. N-field) and, if applicable, the generation of the modifier portion (i.e. M-field) of the instruction word.

DATA STATEMENTS

In the case of constant data statements this field contains the items of data, written left to right. For variable data statements it contains symbols of up to eleven alphanumeric characters (five for PLAN 1), which name the locations that require to be allocated. The number of items that appear on one line is governed only by the length of the field, 57 columns.

The operand field may contain any statement specified in the section in this chapter dealing with formats of data statements (see page 8 et. seq.)

DIRECTIVES

The operand field may contain any parameters specified under the description of the format of the various directives in Chapter 6 of this manual.

COMMENT

Comment may additionally be written in the operand field of any of the three types of statement. The comment must be preceded by a left-hand square bracket, i.e. [. In PLAN 1 there must be a space before the bracket. Where possible, the square bracket is conventionally written in column 36 of the coding sheet with the comment immediately following.

The square bracket is not required if the statement is a comment directive statement. This is indicated by the directive symbol # in column 1, with the remainder of the label, operation and accumulator fields left blank. In this case the comment may start in the first column of the operand field.

Comments will be listed in the compiler printout, but will have no effect on the object program.
The Program Identity Field

The program identity field is intended for use when the source program is punched into cards. It can be used to contain a code number that uniquely identifies the program to which a card belongs.

The Sequence Field

The sequence field is also intended for use when the source program is punched into cards. The cards in the source program pack may be numbered sequentially, the number being written in the sequence field.
FORMATS OF DATA STATEMENTS

Data statements in PLAN 2, 3 and 4 are made under the control of a #LOWER or #UPPER directive. Data statements are either variable or constant. Variable data statements made under a #LOWER or #UPPER directive cause storage to be allocated in the area of lower or upper memory assigned by the compiler to variable data. Constant data statements made under a #LOWER or #UPPER directive cause specified values to be inserted in locations in the area of lower or upper memory assigned by the compiler to preset data.

Data statements are in the same group if they have been made under the same #LOWER or #UPPER directive.

It is not possible to mix constant and variable data statements on the same line. The first data statement on a line determines the type of line.

It is possible to have constant and variable data statements on separate lines under the same #LOWER or #UPPER directive, in which case the compiler will assign the entire group to the appropriate preset area of store.

Constant data statements may be written in the program area, i.e. under the control of a #PROGRAM directive in which case they will be intermingled with the program instructions. PLAN 3 and PLAN 4 compilers also permit variable data statements to be written in the program area.

PLAN 1

In PLAN 1, variable data statements are always assigned to the non-preset area of the store which precedes the preset area, and so will always be separated from constant data statements in the object program. The #LOWER directive may be used with PLAN 1 according to the rules described in Chapter 6 if it is desired to achieve a store layout which is compatible with that produced by a PLAN 2, 3 or 4 compiler. (See "Note on PLAN 1 Store Layout" on page 3 of this chapter.)

The Form of a Variable Data Statement

The names of the storage areas required are written in the operand field. The label, operation and accumulator fields are left blank. The area names may be up to eleven characters long (or five characters in the case of PLAN 1) and must be unique within a segment. The first character of the name must be alphabetic and the remainder may be alphabetic or numeric.

Each statement may specify more than one area. Each area name is separated from the next by a comma. A comma is implied between lines so that the last name in a statement or line must not be followed by a comma.

The initial value of variable data is always zero.

These points are illustrated in the following example:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>10</th>
<th>11</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>OPERAND</th>
<th>PROD</th>
</tr>
</thead>
<tbody>
<tr>
<td>#LOWER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VEL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CITY, AC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>EL, SPEED</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A1, A2, A3</td>
<td>BUFFER</td>
</tr>
</tbody>
</table>

Areas are allocated storage consecutively in the order specified under the #LOWER or #UPPER directive, the last area of one variable data statement being followed immediately by the first of the next.

Each area will be allocated one storage location unless specified otherwise. The size required for an area is expressed in parentheses following the symbolic name. The expression in parentheses must be absolute, i.e. capable of final evaluation as a number (which must be positive).

See Chapter 6, #DEFINE (page 8) for the complete list of rules for writing absolute expressions.

The three simplest types of absolute expression are:

1. An unsigned decimal integer.
2. An unsigned octal integer.

Chapter 2
3 An absolute symbol (i.e. one previously defined or set equal to an absolute expression under a #DEFINE or #SET directive).

It is possible to give a location two or more names. This is achieved by writing an = symbol between the names required. The size of the area is expressed in parentheses following the last name assigned.

**EXAMPLE**

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>OPERAND</th>
<th></th>
<th>PROD.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#LOWER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BUFFER=PRINTOUT=PRINTFILE(256)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**The Form of a Constant Data Statement**

When a data location is to contain a specific value, the name of the data location is written in the label field and the value to be assigned to it is written in the operand field. The name may be up to eleven characters long or five characters in the case of PLAN I. The first character must be alphabetic and the remainder may be alphabetic or numeric. Several constants may be specified separated by commas, on the same line as the label, and on succeeding lines. This will cause successive locations to be set to the values specified. The last value specified on any line of a constant data statement must not be followed by a comma.

A comma written directly after another comma will result in a zero being generated. If a comma is written after the last value on a line a zero word will be generated after the last value.

A value defined in the operand field of a constant data statement may take one of the following forms:

1. An unsigned octal integer.
2. A decimal integer.
3. A decimal fraction.
4. A mixed decimal number.
5. A floating-point number.
6. An unsigned sterling constant.
7. A string of six-bit characters.
8. An index word.
9. A character index word.

An operation statement may also be defined as constant data. The operation statement must be written in the label, operation, accumulator and operand fields in the normal way.

Each type of constant data statement is described below.

**UNSIGNED OCTAL INTEGERS**

The entry in the operand field may be in the inclusive range #0 to #77777777. The octal integer is preceded by the character `#`.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>OPERAND</th>
<th></th>
<th>PROD.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#17, #10.70.70.70.#176.#136274002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SIGNED OR UNSIGNED DECIMAL INTEGERS**

The entry in the operand field may be in the inclusive range

1. -8388608 to +8388607 (single length),
2. -70368744177664 to +70368744177663 (double-length).

Single-length integers are merely written out as data items, or can be followed by (1).

Double-length integers must be followed by (2) to indicate that two words are required to hold the number.

Unsigned integers are assumed to be positive.
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that the largest negative number which may be held in one or in two computer words (-2\(^{23}\) and -2\(^{46}\) respectively) cannot be preset as a decimal integer. If these constants are required, they may be written in some other way, for example:

\[
256/0 \quad \text{for} \ -8388608
\]
\[
\#40000000, 0 \quad \text{for} \ -70368744177664(2)
\]

**SIGNED OR UNSIGNED DECIMAL FRACTIONS**

The entry in the operand field may be in the inclusive range

1. \(-1.0 \text{ to } +1.0 - 2^{23}\) (single-length),
2. \(-1.0 \text{ to } +1.0 - 2^{46}\) (double-length),

Double-length fractions must be followed by (2) to indicate that two words are required to hold the number.

Unsigned fractions are assumed to be positive. Zero is permitted before the decimal point.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The decimal point must be present. A maximum of 14 digits written after the decimal point will be taken into consideration in calculating the fraction. In the case of single-length fractions the value in the word holding the more significant half of the result will be rounded up (if B1 of the second word equals 1) to give the most accurate possible single-length binary equivalent of the decimal fraction.

For double-length fractions the result will be the most accurate possible double-length binary equivalent of the decimal fraction (up to the maximum of 14 digits).

**SIGNED OR UNSIGNED MIXED DECIMAL NUMBERS**

The entry in the operand field may be any combination of integer and fractions (separated by a decimal point) permitted by the expression \((a, b)\)

where \(a = \) the number of words the mixed number will occupy (either one or two),

\(b = \) the number of bits representing the fractional part (in the inclusive range 0 to 46 for double-length, 0 to 23 for single-length).

An expression in the form \((a, b)\) must follow each mixed decimal number specified.

'\(a\)' must be present and have a value in the permitted range. '\(b\)' must be present but may take the value 0.

An unsigned mixed number is assumed to be positive.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If either the integer or fraction is zero, then it may be omitted.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SIGNED OR UNSIGNED FLOATING-POINT NUMBERS (for PLAN 1, 2, 3 and compiler #XPLN)

The entry in the operand field takes the form

\[ \text{AED} \]

where

- \( A \) = the argument, a signed or unsigned decimal fraction in the range \(-9999999999999 \) to \(+9999999999999\)
- \( D \) = the exponent, a signed or unsigned decimal integer in the range \(-76 \) to \(+76\).

The two parts are separated by the character \( E \).

The numerical value of a non-zero floating-point number appearing in a constant data statement should not be less than \( 10^{-76} \) nor greater than \( 10^{76} \).

Each generated constant is converted to normalized binary floating-point and stored as a double-length floating-point number in two consecutive locations. An unsigned argument or exponent is assumed to be positive.

SIGNED OR UNSIGNED FLOATING-POINT NUMBERS (for PLAN 4 with #XPLT)

#XPLT will accept data statements in the form

\[ \text{S.T.E.Q} \]

for floating-point numbers

\[ \text{S.T.D.Q} \]

for extended precision floating-point numbers

where

- \( S \) = the argument
- \( T \) = the exponent
- \( Q \) = the exponent

Only 27 digits of \( S.T \) are significant although more than that number will be accepted. This gives a floating-point data statement having an approximate range, in decimal, of

\[ 10^{-76} \lt A \lt 10^{76} \]

and the argument of an extended precision floating-point number is accurate to within \( 2^{-74} \).

The compiler considers data statements in the following way:

The data statement \( \text{abcd.efghij} \{ \begin{array}{c} D \\ E \end{array} \} \), where \( \text{abcd.efghij} \) are all integers considered as being \( \text{abcd.efghij} \times 10^{q+4} \).

\( \text{abcd.efghij} \) is evaluated as a four word binary fraction \( F \). \( Q \) is evaluated and adjusted to give \( q+4 \) according to the position of the decimal point in the argument (thus, \( q' = q+4 \)). \( 10^{q} \) is evaluated to give a normalized four word fraction \( Z \) and an exponent \( W \) such that \( 10^{q} = Z \cdot 2^{W} \). The two fractions \( F \) and \( Z \) are multiplied and normalized and \( W \) is adjusted accordingly to give a result \( X \). \( X \) and \( W \) are then used to give the extended precision constant which may be further shortened to produce a normal floating-point constant.

UNSIGNED STERLING CONSTANTS

The entry in the operand field may be in the range

1. 0d to £34,952/10/7 (single-length),
2. 0d to £293,203,100,740/5/3 (double-length).

Sterling constants are identified by the leading character, which must always be the £ symbol. The constituent parts of the word, i.e. pounds, shillings and pence, must be separated by oblique strokes, i.e. /. If the pounds, shillings or pence fields are zero then they should be represented by a zero character, e.g. £0/10/0. Double-length sterling constants must be followed by (2). 10d and 11d must be punched as two characters.
The compiler will automatically convert sterling constants to binary pence before storing them. It is permissible to indicate any number of shillings or pence (up to the equivalent of the limits stated above), but if the value of shillings exceeds 19 or the number of pence exceeds 11, any higher field should have an entry of zero. For example, if it was required to store £18/5/0 in store, it could be written as £0/365/0. On the other hand, if £18/5/0 was written as £1/345/0, it would be flagged as an error.

If decimal places of pence are required, they must be preceded by a decimal point and the constant must be followed by an expression of the form \((a \cdot b)\), where \(a\) is the number of words required to hold the mixed number (representing binary pence), and \(b\) is the number of binary digits that will represent the fractional part. For example:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
<th>72</th>
<th>76</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>398/16/6/75 (2/2.3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CHARACTERS

The entry in the operand field takes the form

\[ n \text{HCHARACTERS} \]

where \(n\) = the number of characters following,

H = an indication to the compiler that the following symbols are to be inserted in a word in six-bit character form,

CAHRACETERS = any combination of Characters in the ICL 64-Character Card Code (see Appendix 1).

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
<th>72</th>
<th>76</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>1HL 12NXY 8KH39 0 1 2HAL 3KH6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROSS</td>
<td>42HG6OUTDPORTERSPORTEENDENPATHSAYS13RAVAT7AYVRUMASBRANDYCHAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each item will occupy an integral number of words with spaces being filled to the right. A group of up to 52 characters may be defined on one line. If more than one line is required to hold the characters, each line, except possibly the last, must have a multiple of four characters since unused character positions in the last word used will be space filled. Each line must begin with \(n\)H, where \(n\) represents the number of characters in the line as explained above.

Note: If paper tape is being used for input and any of the four characters $, ], ↑ and ← are among the characters defined, then the maximum number of characters that may be defined on one line is reduced by one for each appearance of any of those four characters on that line.

INDEX WORDS

Index items will generate words that are split into a 9-bit counter and 15-bit modifier. The counter and the modifier parts of the item are separated by a solidus (/) which must be present. The entry in the operand field takes the form

\[ \text{COUNT/MODIFIER} \]

where COUNT may be

1. an unsigned decimal integer in the range 0 to 511, or
2. an octal integer in the range #0 to #777, or
3. a symbol previously defined by a #DEFINE or #SET directive, to be equal to a value in the range 0 to 511, and MODIFIER may be

1. a previously defined symbol which refers to a storage location in lower or upper data,
2. an octal or decimal integer in the range 0 to 32,767,
3. an undefined symbol which will be treated as a forward reference or branch ahead and which may subsequently be defined by its appearance in the label field of a program instruction written under the control of a #PROGRAM directive,
4. a symbol as in 1 or 3 above followed by a signed octal or decimal integer.
any absolute or relative expression which does not use multiplication or division. 2 and 4 are simple examples of absolute and relative expressions respectively. See Chapter 6, page 8 for the complete rules for writing these expressions.

CHARACTER INDEX WORDS

Character index items will generate words that have a seven-bit counter, while the modifier includes two extra bits to specify a character position. Thus, they are the same as index words with the following two exceptions:
1. The value of the counter must be in the range 0 to 127.
2. The modifier part terminates with n, where n is one of the digits 0, 1, 2, or 3.

INSTRUCTION STATEMENTS AS CONSTANT DATA

Any valid instruction statement is acceptable as a statement of constant data made under the control of a #LOWER or #UPPER directive. It must be written on a line separate from any other statements, in the exact form required. It will be evaluated and stored in sequence with the other constants. It may, but need not, be labelled.

Consider this example:

Location DATA will contain the binary form of the machine code instruction generated by the PLAN statement

GENERAL FORM OF AN OPERATION STATEMENT

A PLAN instruction consists of three basic parts, the Operation, the Accumulator and the Operand. Each is written, left-justified, in the appropriate field of the coding sheet. (In PLAN 1 only, the first space in any field is taken as terminating that field.) Any instruction statement may optionally be given a label.

The general form (ignoring the optional label) may be summarized symbolically as:

where F is the Operation Mnemonic,
X is the Accumulator,
N is the Operand,
M is the Modifier (if any).

The formats for the label, operation and accumulator fields are given in the appropriate sections on the PLAN coding sheet (page 4) and details of the type of operand which are permissible in operation statements follow.

Certain conventions apply in this manual to the use of the above symbols for the accumulator and operand fields. Thus, for the accumulator field, X will indicate one and only one accumulator to be specified when writing these
instruction statements. XX* will indicate a double-length operation, and will require that two accumulators, X and (X + 1), be specified. In certain PLAN 4 statements X, Xm will indicate that two accumulators (one to hold a counter, the other to hold a modifier) must be specified but that the two accumulators need not be adjacent. If an operation statement has an M-field then the operand may be followed by (M), where M is one of the digits 1, 2, or 3 giving the address of the index register (or modifier) to be used when executing the instruction. For the operand field, the use of N(M) indicates that the instruction has a twelve-bit operand plus a two-bit modifier portion.

When an instruction is shown with only N in the operand field, then this instruction has a 15-bit address.

In the operand field, leaving the field blank is equivalent to writing 0. This is not so in the accumulator field, however, because 0 is an accumulator address, and in those instructions for which this field is specified as blank the effect of writing 0 is undefined.

TYPES OF OPERAND IN OPERATION STATEMENTS

The basic type of instruction in PLAN operates on a word of data storage whose address is given in the operand field of the instruction. The operand may be quoted as a number or a symbol, but in either case will be evaluated by the compiler as a number. This number is in the range 0 to 4095 (and may take a value up to 32,767 in compact mode, or 4,194,303 in extended data mode, if modified when the instruction is obeyed) in the case of non-branch instructions, or 0 to 32,767 in the case of branch instructions. Operands may be expressed as absolute expressions or relative expressions. These terms are explained in the following pages.

Some groups of non-branch instructions operate not on a data storage word but on the contents of the operand itself as held in the program operation store. A direct operand of this type may again be either numeric or symbolic and will be evaluated as a number in the range 0 to 4095 (and may take a value up to 32,767 in compact mode, or 4,194,303 in extended data mode, if modified when the instruction is obeyed). This number is not an address but is the numeric value to be used by the instruction. In the case of symbolic data location names used as direct operands, the address of the location specified is used as the numeric value.

There is a method of achieving the effect of a direct operand with a normal data addressing instruction. This is done by enclosing the operand in single inverted commas. Such an operand is called a literal, and it will cause the compiler to set up and reference in lower data a word containing the value quoted in the operand. Literal operands are not modifiable.

Another form of operand, used with branch instructions, permits an address to be defined in terms of a number added to or subtracted from the address of the current instruction. This form is known as a relative operand.

Considering operand formats in PLAN operation statements, and in particular the possible forms of N, four main types of operand can be distinguished:

1. Absolute expressions.
2. Relative expressions.
3. Literal operands.
4. Relative operands.

Details of how to write each of these types are given below and they are examined as applicable to

1. branch instructions, and
2. non-branch instructions.

The instructions included in each group referred to are given in the summary table of operations, on pages 21 and 22. Special cases are not considered in this chapter but are covered in Chapter 4 in which details are given for each operation statement individually.
Rules for Writing Absolute Expressions in the Operand Field of Instruction Statements

An absolute expression is one which can be evaluated as a number independent of any actual location address occupied by any word of data or program instruction.

An absolute expression may consist of any number of decimal integers, octal integers and symbols.

Each integer or symbol must be preceded by a plus or minus sign, except in the case of the first term of an expression for which (in the absence of a sign) a plus sign is assumed. The expression is evaluated serially from left to right.

If a symbol is used in the expression, it must either

1. be associated somewhere in the expression with a symbol of the same group and be preceded by the opposite sign to this accompanying symbol (i.e. the form must be \( A - B \) or \(-A + B\), where \( A \) and \( B \) represent two symbolic identifiers of the same group), or

2. have been set equal to an absolute expression in some previous directive.

In the case of 1 above symbols are considered to be in the same group if they have been defined under the same \#LOWER or \#UPPER directive, or if they both appear in the label field of the program area, in the same segment.

Rule 1 is due to the fact that the programmer does not directly control the location addresses of words of variable, preset or program areas but he can regard the difference between the addresses of two symbols of the same group as a fixed (absolute) quantity.

All symbols appearing in an absolute expression must have been previously defined.

The three simplest forms of absolute expression used in operand fields are:

1. A decimal integer.
2. An octal integer.
3. An absolute symbol, i.e. one that has been previously defined by \#DEFINE or \#SET as an absolute expression. See Chapter 6, \#DEFINE.
Absolute Expressions in Operand Fields

BRANCH INSTRUCTIONS
Not used.

NON-BRANCH INSTRUCTIONS

1. The operand N(M) is a direct operand in the following groups:
   (a) Arithmetic with small integers.
   (b) Logical operations with small integers.

2. The operand N(M) is not a location but the actual value used in the operations for the following groups:
   (a) Shift instructions.
   (b) Floating-point normalize.
   (c) Block transfer and check sum.
   (d) Some of the miscellaneous group, e.g. MODE.

Rules for Writing Relative Expressions in the Operand Field of Instruction Statements

A relative expression depends for its ultimate evaluation upon the actual address occupied by some word of data or program instruction.

A relative expression contains one positive non-absolute symbol (i.e. a symbol referring to a location) which does not conform with rules 1 or 2 given above for writing absolute expressions, but is otherwise absolute.

The two simplest forms of relative expression used in operand fields are:

1. A symbolic name.

2. A symbolic name adjusted by following it with a signed octal or decimal integer.

In the second case, if the adjusting integer is negative, it must not be big enough to take the resulting address below that of the first location of the area in which the symbolically named word is located. For example, say that the following had appeared on a coding sheet:

   #LOWER
   WEST(5), EAST(10)

A relative expression such as "EAST-4" could subsequently be used in the operand field where appropriate; but an attempt to use, say, "EAST-6" or "WEST-1" would be flagged by the compiler as a J class error (see Chapter 9).

Relative Expressions in Operand Fields

BRANCH INSTRUCTIONS

The operand N refers to a location in the program operation store.

If the relative expression is a symbolic name and the symbol is undefined it is assumed by the compiler to be a forward reference.

NON-BRANCH INSTRUCTIONS

The operand N(M) takes the same forms as for absolute expressions used with non-branch instructions.

If the relative expression is a symbolic name and the symbol is undefined, the compiler will allocate a location in lower data to the symbol and the value of the symbol will be the address of this location (not PLAN 1). In the case of a double-length operation two locations are allocated and the symbol is given the value of the first location.

NOTE: The rules for writing absolute and relative expressions given here are the same as those given with #DEFINE (Chapter 6) except that multiplication and division operations are excluded from the expressions.
Literal Operands

BRANCH INSTRUCTIONS

Not used.

NON-BRANCH INSTRUCTIONS

Literal operands are not permitted with the following groups of instructions:
1. Those groups that leave their results in locations specified by the operand, e.g. arithmetic into store.
2. Floating-point instructions.
3. Arithmetic or logical operations with direct operands.
4. Any double-length operation.

Literal operands are not modifiable. They cannot be used in PLAN 1.

The literal operand, used with instruction statements, is another method of data generation and is recognized by a beginning and ending quote mark.

In general, the literal can take any of the forms specifiable in a data statement except that all literals result in only a single word of data, and that floating-point numbers are not permitted. Although all literals are thus single-length, literals which refer to locations in common areas result in the generation of a second word which is used by the compiler during assembly, and this word is also inserted in the object program.

When the compiler encounters an instruction with a literal operand, it will first evaluate the data within the quote marks to form a binary number. If the binary number is the same as that produced by some prior literal, then the instruction address portion is given a value equal to the location already assigned to the binary number. If the binary number is unique, then it is assigned a location in lower memory, and the instruction address portion is given a value equal to this assigned location.

The operand may be:

1. A signed or unsigned decimal integer in the range \(-8388608\) to \(+8388607\). The same as for data statements, limited to single precision.

   An unsigned integer is assumed to be positive.

2. A signed or unsigned decimal fraction in the range \(-1.0\) to \(+1.0 - 2^{-23}\). The same as for data statements, limited to single precision.

   An unsigned fraction is assumed to be positive.

3. A signed or unsigned mixed number. The same as for data statements, except that the number of words the mixed number will occupy is limited to one.

   An unsigned mixed number is assumed to be positive.
If either the integer or fraction is zero, it may be omitted.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Acc</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LDX</td>
<td>4</td>
<td>'2001.06'</td>
</tr>
<tr>
<td>2</td>
<td>LDX</td>
<td>4</td>
<td>'678.321'</td>
</tr>
</tbody>
</table>

4 An octal integer in the range #0 to #7777777. The same as for data statements.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Acc</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LDX</td>
<td>4</td>
<td>'#202020'</td>
</tr>
<tr>
<td>2</td>
<td>LDX</td>
<td>4</td>
<td>'#767676'</td>
</tr>
</tbody>
</table>

5 Up to four 6-bit characters. The same as for data statements, but limited to a maximum of four characters.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Acc</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LDX</td>
<td>4</td>
<td>'YES'</td>
</tr>
<tr>
<td>2</td>
<td>LDX</td>
<td>4</td>
<td>'HNY'</td>
</tr>
</tbody>
</table>

6 An index word. The same as for data statements except that if the counter is not present the solidus may be omitted.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Acc</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LDX</td>
<td>3</td>
<td>'/ANNE'</td>
</tr>
<tr>
<td>2</td>
<td>LDX</td>
<td>3</td>
<td>'/HECH+10'</td>
</tr>
<tr>
<td>3</td>
<td>LDX</td>
<td>3</td>
<td>'/NAME'</td>
</tr>
</tbody>
</table>

7 A character index word. The same as for data statements but the solidus may be omitted if the counter is not present unless the modifier is 0 (for example, the ambiguous operand '0.2' is taken by the compiler to be a fraction).

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Acc</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LDX</td>
<td>4</td>
<td>'/PRINT:3'</td>
</tr>
<tr>
<td>2</td>
<td>LDX</td>
<td>4</td>
<td>'/CATH+1.3'</td>
</tr>
<tr>
<td>3</td>
<td>LDX</td>
<td>4</td>
<td>'/TABLE:1'</td>
</tr>
</tbody>
</table>

8 A data area name

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Acc</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LDX</td>
<td>3</td>
<td>'HENER'</td>
</tr>
</tbody>
</table>

If the name of a location is quoted as a literal operand, the compiler will cause a constant equal to the address of the location to be stored in lower data and will create a reference to the constant in the compiled instruction. This facility is principally used to access upper data locations. If an undefined symbol is quoted it will be treated as a branch ahead.

Relative Operands

**BRANCH INSTRUCTIONS**

The entry is the operand field takes the form:

* ± N

where N is an integer which is added to, or subtracted from, the address of the branch instruction, to give the instruction that is to be obeyed next.
The relative operand applies to instructions as they appear in the machine-code object program, and not to instructions written in the source program. If macro-instructions are used special care must be exercised in the use of the relative operand. For example:

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>AC+</th>
<th>RC</th>
<th>RC+</th>
<th>RC+</th>
<th>RC+</th>
<th>RC+</th>
<th>RC+</th>
<th>RC+</th>
<th>RC+</th>
<th>RC+</th>
<th>RC+</th>
<th>RC+</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX</td>
<td>A5, Y7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this example, though there are only two PLAN instructions, three machine-code instructions are generated, so to jump from BDN to CALL requires a relative operand * + 3 (and not * + 2).

Since an instruction within a macro-instruction cannot be labelled, any branch instructions in it which jump to instructions also within the macro must use relative operands.

NON-BRANCH INSTRUCTIONS

Not used.

VERSIONS OF PLAN - SUMMARY OF FACILITIES

PLAN 1

The facilities available in PLAN 1 are:

1. Use of mnemonic functions, for those instructions which are available (macros and certain other instructions are excluded).
2. Use of symbolic names for addresses, provided that explicit definition has preceded the use of each name in the program.
3. As necessary, basic directives for controlling the compilation of the program, including various methods for defining constants.
4. The incorporation of library subroutines at compilation time (during the consolidation run).
5. The incorporation of Cassette Tape Housekeeping routines in a similar manner.

PLAN 2

Facilities available in PLAN 2, in addition to those in PLAN 1 are:

1. The use of 'literal operands' to avoid having to define constants separately.
2. The implicit definition of names for variables, by their use in the program.
3. Additional directives to make compilation more flexible.
4. Extension of the data area to use that part of the program beyond the first 4K, which is not accessible by direct addressing (UPPER DATA).
5. Specification of COMMON storage to facilitate program segmentation.
6. Use of the LDPL operation.

PLAN 3

Facilities available in PLAN 3, in addition to those in PLAN 1 and 2 are:

1. Incorporation of the Input/Output Generator, Magnetic Tape Housekeeping and Direct Access Housekeeping routines.
2. Writing of storage device compatible programs.
3. Incorporation of dump and restart routines.
4. Macro-instructions.
Subprogramming.
Overlay programming.
Additional directives.

PLAN 4
Facilities available in PLAN 4, in addition to those available in PLAN 1, 2 and 3, are:
1 Facilities for specifying the address mode in which the program is to operate initially.
2 Facilities for specifying the address mode compatibility of the component segments of a program.
3 Facilities for generating 22-bit modifiers, and further monitoring of 15-bit modifiers.
4 Additional macro-instructions to facilitate the handling of two-word counter-modifiers in extended data mode.
5 With PLAN 4 compiler #XPLT, facilities for using extended precision floating-point numbers.
## SUMMARY OF OPERATIONS

<table>
<thead>
<tr>
<th>Arithmetic into Accumulators</th>
<th>Block Transfer and Check Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX X N(M) Load into X</td>
<td>MOVE X N(M) Move N words</td>
</tr>
<tr>
<td>ADX X N(M) Add into X</td>
<td>SUM X N(M) Sum N words</td>
</tr>
<tr>
<td>NGX X N(M) Negate into X</td>
<td></td>
</tr>
<tr>
<td>SBX X N(M) Subtract from X</td>
<td>Comparison</td>
</tr>
<tr>
<td>LDXC X N(M) Load into X and set carry if appropriate</td>
<td>TXU X N(M) Test contents of X unequal to contents of N(M)</td>
</tr>
<tr>
<td>ADXC X N(M) Add to X and set carry if appropriate</td>
<td>TXL X N(M) Test contents of X less than contents of N(M)</td>
</tr>
<tr>
<td>NGXC X N(M) Negate into X and set carry if appropriate</td>
<td></td>
</tr>
<tr>
<td>SBXC X N(M) Subtract from X and set carry if appropriate</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arithmetic into Store</th>
<th>Branch on State of Accumulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>STO X N(M) Store</td>
<td>BZE X N Branch if zero</td>
</tr>
<tr>
<td>ADS X N(M) Add to store</td>
<td>BNZ X N Branch if non-zero</td>
</tr>
<tr>
<td>NGS X N(M) Negate into store</td>
<td>BPZ X N Branch if positive</td>
</tr>
<tr>
<td>SBS X N(M) Subtract from store</td>
<td>BNG X N Branch if negative</td>
</tr>
<tr>
<td>STOC X N(M) Store and set carry if appropriate</td>
<td></td>
</tr>
<tr>
<td>ADSN X N(M) Add to store and set carry if appropriate</td>
<td></td>
</tr>
<tr>
<td>NGSC X N(M) Negate into store and set carry if appropriate</td>
<td></td>
</tr>
<tr>
<td>SBSC X N(M) Subtract from store and set carry if appropriate</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arithmetic with Small Integers</th>
<th>Unconditional Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDN X N(M) Load N into X</td>
<td>BRN N Branch to N</td>
</tr>
<tr>
<td>ADN X N(M) Add N to X</td>
<td></td>
</tr>
<tr>
<td>NGN X N(M) Negate N into X</td>
<td></td>
</tr>
<tr>
<td>SBN X N(M) Subtract N from X</td>
<td></td>
</tr>
<tr>
<td>LDNC X N(M) Load N and set carry if appropriate</td>
<td>Indexing and Counting</td>
</tr>
<tr>
<td>ADNC X N(M) Add N and set carry if appropriate</td>
<td>BUX X N Branch on unit indexing</td>
</tr>
<tr>
<td>NGNC X N(M) Negate N and set carry if appropriate</td>
<td>BDX X N Branch on double indexing</td>
</tr>
<tr>
<td>SBNC X N(M) Subtract N and set carry if appropriate</td>
<td>BCHX X N Branch on character indexing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplication</th>
<th>Subroutine Linkage</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPY X N(M) Multiply</td>
<td>CALL X N Call subroutine at N</td>
</tr>
<tr>
<td>MPR X N(M) Multiply and Round</td>
<td>EXIT X N Exit to N + X</td>
</tr>
<tr>
<td>MPA X N(M) Multiply and Accumulate</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Division</th>
<th>Subprogramming Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVD X N(M) Divide</td>
<td>SUSMA X N(M)</td>
</tr>
<tr>
<td>DVR X N(M) Divide and Round</td>
<td>AUTO X N(M)</td>
</tr>
<tr>
<td>DVS X N(M) Divide single-length</td>
<td>SUSAR X</td>
</tr>
<tr>
<td>Character Conversion</td>
<td>SUSIN X</td>
</tr>
<tr>
<td>CDB X N(M) Decimal to binary</td>
<td></td>
</tr>
<tr>
<td>CBD X N(M) Binary to decimal</td>
<td></td>
</tr>
</tbody>
</table>
### SUMMARY OF OPERATIONS continued

<table>
<thead>
<tr>
<th>Shifting</th>
<th>Floating-point Arithmetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLC X N(M) Shift left cyclic</td>
<td>FAD X N(M) Add to A</td>
</tr>
<tr>
<td>SLL X N(M) Shift left logical</td>
<td>FSB X N(M) Subtract from A</td>
</tr>
<tr>
<td>SLA X N(M) Shift left arithmetic</td>
<td>FMPY X N(M) Multiply</td>
</tr>
<tr>
<td>SLC XX* N(M) Shift left cyclic (double)</td>
<td>FDVD X N(M) Divide into A</td>
</tr>
<tr>
<td>SLL XX* N(M) Shift left logical (double)</td>
<td>LFP N(M) Load A</td>
</tr>
<tr>
<td>SLA XX* N(M) Shift left arithmetic (double)</td>
<td>SFP N(M) Store A</td>
</tr>
<tr>
<td>SRC X N(M) Shift right cyclic</td>
<td>LFPZ N(M) Zeroize A</td>
</tr>
<tr>
<td>SRL X N(M) Shift right logical</td>
<td>SFPZ N(M) Store A and zeroize</td>
</tr>
<tr>
<td>SRA X N(M) Shift right arithmetic</td>
<td>SRAV XX* N(M) Shift right arithmetic on overflow (double)</td>
</tr>
<tr>
<td>SRAV X N(M) Shift right arithmetic on overflow</td>
<td></td>
</tr>
<tr>
<td>Logical Operations into Accumulator</td>
<td>Floating-point Conversion</td>
</tr>
<tr>
<td>ANDX X N(M) AND to X</td>
<td>FLOAT N(M) Fixed to floating</td>
</tr>
<tr>
<td>ORX X N(M) OR to X (inclusive OR)</td>
<td>FIX N(M) Floating to fixed</td>
</tr>
<tr>
<td>ERX X N(M) Exclusive OR to X</td>
<td></td>
</tr>
<tr>
<td>Logical Operations into Store</td>
<td>Peripheral Control</td>
</tr>
<tr>
<td>ANDS X N(M) AND to store</td>
<td>SUSBY X N(M) Suspend if busy</td>
</tr>
<tr>
<td>ORS X N(M) OR to store (inclusive OR)</td>
<td>REL X N(M) Release</td>
</tr>
<tr>
<td>ERS X N(M) Exclusive OR to store</td>
<td>DIS X N(M) Disengage</td>
</tr>
<tr>
<td>Logical Operations with Small Integers</td>
<td>ALLOT X N(M) Allocate</td>
</tr>
<tr>
<td>ANDN X N(M) AND with N</td>
<td>PERI X N(M) Peripheral transfer</td>
</tr>
<tr>
<td>ORN X N(M) OR with N (inclusive OR)</td>
<td></td>
</tr>
<tr>
<td>ERN X N(M) Exclusive OR with N</td>
<td></td>
</tr>
<tr>
<td>Part Word Manipulation into Accumulators</td>
<td>Interrupt and Delete</td>
</tr>
<tr>
<td>LDCH X N(M) Load character into X</td>
<td>SUSTY N(M) Suspend and type</td>
</tr>
<tr>
<td>LDEX X N(M) Load exponent into X</td>
<td>DISTY N(M) Display message</td>
</tr>
<tr>
<td></td>
<td>DELTY N(M) Delete and type</td>
</tr>
<tr>
<td></td>
<td>SUSWT N(M) Suspend and wait</td>
</tr>
<tr>
<td></td>
<td>DISF N(M) Display</td>
</tr>
<tr>
<td></td>
<td>DEL N(M) Delete</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Extended Floating-point Operations</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>LEFP N(M) Load extended floating-point</td>
<td>OBEY N(M) Obey the instruction in N</td>
</tr>
<tr>
<td>LAEZ N(M) Load accumulator extension</td>
<td>STOZ N(M) Store zero</td>
</tr>
<tr>
<td></td>
<td>zero</td>
</tr>
<tr>
<td>RFP N(M) Round floating-point</td>
<td>NULL N(M) Null operation</td>
</tr>
<tr>
<td>SAE N(M) Store accumulator extension</td>
<td>LDCT X N(M) Load into counter</td>
</tr>
<tr>
<td>SAEZ N(M) Store accumulator extension</td>
<td>MODE N(M) Set zero suppression mode</td>
</tr>
<tr>
<td>and zeroize</td>
<td>CONT X N(M) Read in more program</td>
</tr>
<tr>
<td>NFPS N(M) Negate floating-point</td>
<td>RQX X N(M) Read or replace request</td>
</tr>
<tr>
<td>accumulator and store</td>
<td>block</td>
</tr>
<tr>
<td>ZFPS N(M) Zeroize floating-point</td>
<td>SUSDP X N(M) Suspend and dump</td>
</tr>
<tr>
<td>accumulator and store</td>
<td>GIVE X N(M) Give information or change</td>
</tr>
<tr>
<td></td>
<td>the internal operating</td>
</tr>
<tr>
<td></td>
<td>environment</td>
</tr>
<tr>
<td></td>
<td>Miscellaneous, available on 1902A, 1903A and</td>
</tr>
<tr>
<td></td>
<td>Processors with Extended Data Mode facilities.</td>
</tr>
<tr>
<td></td>
<td>BCT N Branch on count</td>
</tr>
<tr>
<td></td>
<td>BFP X N Branch on floating-point</td>
</tr>
<tr>
<td></td>
<td>accumulator</td>
</tr>
<tr>
<td></td>
<td>MVCH X N(M) Transfer N characters</td>
</tr>
<tr>
<td></td>
<td>SMO N(M) Supplementary modifier</td>
</tr>
</tbody>
</table>
Chapter 3 Additional Facilities

INTRODUCTION
This chapter deals with some of the special PLAN facilities. There are separate sections on subprogramming, program segmentation, overlay programming and programming for paged environments.

SUBPROGRAMMING
A feature of PLAN 3 and 4 is a subprogramming facility which allows programs to be divided into separate subprograms or members with controlled intercommunication. This enables processes which are to some extent independent to be time-shared with each other. The time-sharing operates in a manner similar to Executive time-sharing between programs. The division of a program for subprogramming purposes is achieved by specifying the relevant entry in the operand field of a #PROGRAM directive (see Chapter 6). The standard PLAN instructions for writing subprograms are AUTO, SUSAR, SUSIN and SUSMA (see Chapter 4 for a specification of these instructions).

In the subprogramming system a program consists of a number of members each of which is equal in status to the other members. This system is most suitable for the control of one or more "interrupt" peripherals, i.e. peripherals whose action is not completely under the control of the program. Programs using this system can be run on any central processor with a suitable Executive, except the 1901 and 1901A.

In the subprogramming system the allocation of central processor time between members, or between master and subprograms, is controlled by Executive according to the priority given to each by the programmer or operator.

PROGRAM SEGMENTATION
When a PLAN program is being written and developed it may be divided into units called segments. Program segmentation enables parts of a program to be written and tested separately. At a subsequent stage the various segments may be consolidated to provide a complete working program.

Segmentation has advantages when programs are being prepared and also when program amendments are being made. It is also possible to combine segments written in different languages. Although a program may be written in segmented form, there should initially be a concept of the program as a whole.

Normally, symbolic names for data locations must be unique within a program, but where segmentation is employed these names need only be unique to any one segment. The symbolic names must, however, be defined separately for each segment. If several segments define a data area with the same name, then separate areas will be allocated for each segment. Instruction labels are also unique only within a segment.

Each segment of a program is headed by a #PROGRAM directive (see Chapter 6) setting out the four-character name of the whole program and the name of the segment, which may be up to eleven alphanumeric characters, of which the first must be alphabetic. This directive will be followed by the data and instructions under #LOWER, #UPPER and #PROGRAM directives in the usual way. The segment is ended by an #END directive (and a #FINISH directive in the last segment of a PLAN 2 or 3 program). A program may consist of a single segment in which case #END may be preceded by a #COMPLETE directive.
Segment Intercommunication in PLAN 2, PLAN 3 and PLAN 4 Programs

Since data and instruction labels generally only have meaning within the segment in which they are written some means is required of achieving communication between segments.

Data communication is carried out through common data areas which are defined by a \#LOWER directive in the form described in Chapter 6. The operand of this directive specifies COMMON and the name to be given to a common block of store. The block then becomes common to all segments and will be referenced by any segment which uses the defined block name. If space allocated to a common block name in one segment differs in size from that allocated in another segment then PLAN will allocate the larger area. The directive \#UPPER may be used in a similar way to define areas in upper storage.

A branch to a program instruction in a different segment is made possible by use of a \#CUE directive. This directive precedes a program instruction and defines for it a cue name which is unique for the whole program, and may therefore be used as a branch address in any other segment.

Subroutines

A subroutine or segment may be entered by quoting its name in the operand field of a CALL instruction. An alternative method is to branch to a cue (see the previous section).

Library subroutines

Incorporation of a library subroutine is normally achieved by writing a CALL instruction at the appropriate point in the source program, with the name of the subroutine in the operand field. In some cases the CALL instruction will be followed by parameters required by the subroutine. Details of how to write these parameters, together with which accumulator to use in the CALL instruction (the link accumulator), are given in Library Specifications for commercial and general library subroutines, and in Scientific Subroutines for scientific library subroutines. The library subroutine will be incorporated into the program during consolidation.

Consolidation of subroutines

If a library subroutine (or other free-standing segment) is CALLED by segment name, it will automatically be sought and incorporated during consolidation.

Consolidation of a free-standing subroutine that is not CALLED by segment name is achieved by means of \#PERMANENT directive (see Chapter 6).

OVERLAY PROGRAMMING

Overlay programming is based on the principle of program segmentation, with the additional feature of holding only a certain number of segments in store at a given time. The core store required by a program may be divided into an area of permanent storage and a number of overlay areas. The permanent storage contains, for the duration of the run, a run-time package (which calls in overlays) and, optionally, other segments of the user program.

The run-time package is provided by ICL, is held on the library as a subroutine, and is inserted during consolidation, by means of a \#PERMANENT directive. The choice of run-time package determines the overlay medium. An overlay program can be transcribed from one medium to another for storage or security purposes using the appropriate standard library manipulation utility program, but when the overlay program is run, it must be held on the medium appropriate to the run-time overlay package which it contains. Each overlay area can hold, at any one time, one of a number of overlay units each of one or more segments. Hence, although a number of overlay units may be present in store at the same time, they must have been specified as being in different areas.

Decisions on overlay organization are not required at the time of writing the source program. Overlay organization and medium is defined at consolidation time by means of a steering segment. This enables a programmer to write segments without reference to their final place in permanent or overlaid program, or to the backing medium which will eventually hold the overlays. For further information on overlays, see Appendix 6; the overlay macro-instructions are described in Chapter 5, pages 55 to 58, and directives for use in the steering segment are described in Chapter 6, pages 19, 26, 29, 32, and 45.
PROGRAMMING FOR PAGED ENVIRONMENTS

A program which is run under GEORGE 4 on a paged processor may incorporate several features which are not available in other environments. It is not within the scope of this manual to describe the GEORGE 4 system; however, those features which may be incorporated in a PLAN program are described here briefly. Further information may be found in the ICL 1900 Series manuals Central Processors and Operating Systems GEORGE 3 and 4 (Edition 4).

The facilities which are available to a programmer are as follows.

Sparse and dense programs

An object program written for non-paged environments has a contiguous virtual store with an addressing range from 0 to the program size -1. Such a program is said to be dense and can be run in a non-paged or a paged environment. In a paged environment the object program does not need to occupy a contiguous area of store, but has a virtual store with an addressing range from 0 to 4096K-1 (although normally addresses of 3072K and above are reserved for use by the consolidation system). Such a program is said to be sparse and can only be run in a paged environment. The instruction GIVE N(M) = 4 is available to enable a programmer to change the addressing range from sparse to dense and vice versa. Note, however, that sparse programs operate in 22AM and EBM and that dense programs may be in any legal combination of operating modes. The reply from the instruction indicates whether the program is sparse or dense and gives the program size, that is, the maximum number of words which the program can currently access.

Shareable programs and areas

In order to optimize on the use of store, areas of one program may be shared by activities of the same program. Such a program is marked by the system as being a shareable program and those areas of the program which may be shared by other activations are marked as being shareable areas. However, unless the whole program is marked as shareable, the shareable areas within that program will not be shared. Segment code may be defined as shareable or non-shareable by giving the appropriate parameter to the segment #PROGRAM directive. A program's data area may be defined as shareable by giving a directive #LOWER or #UPPER. These directives have the same effect as #LOWER and #UPPER. A program may be marked as shareable by giving an appropriate parameter to the consolidator.

It should be noted that shareable areas are also called pure areas, and non-shareable areas are called impure areas. Pure areas are likely to be those areas of a program which remain unchanged during the running of the program and as such will be instruction statements, literals, replacers, presets and any constant data areas.
Chapter 4 Operation Statements

INTRODUCTION
This chapter contains a description of each of the PLAN operation statements. With each operation there is a description of its effect when the program is obeyed. These descriptions cover normal usage of operations, but should not be regarded as complete detailed definitions.

The chapter's layout is described in this section, together with an explanation of the abbreviations that are used throughout.

*The PLAN operation statements described in this chapter are available as PLAN 4 operation statements, except those statements applicable to extended floating point facilities which are available with PLAN 4 #XPLT only.*

LAYOUT OF THE CHAPTER
The PLAN operation statements are described in alphabetical order for purposes of easy reference. Each operation is dealt with under the following headings:

- **Function**  A very brief description of the operation's function.
- **Format**    The contents that are permissible in the operation field, accumulator field and operand field when the operation statement is written in the source program.
  
  This is followed by a sentence which states what the operand (possibly modified) represents in the object program.
  
  The following terms are used in the format description of the operand field:
  
  1. An 'absolute expression'.
  2. A 'relative expression'.
  3. A 'literal'.
  4. A 'relative operand'.
  
  These terms are fully explained in Chapter 2 (page 14 onwards).

- **Execution** A more detailed description of the operation's function and the way in which it affects C or V.

- **Examples** The method of writing and the use of each operation statement is usually illustrated by one or more simple examples.

- **Machine Code Listing** The format of the operation statement as it is listed in the printout.

- **Notes** Any further comments on the operation that may be necessary.
ABBREVIATIONS USED IN THE CHAPTER

The following abbreviations are used in this chapter:

A  Floating-point accumulator
B  Bit or binary digit
C  Carry register

The bit positions within a word are numbered from left to right; the most significant bit is B0 and the least significant bit is B23.

FOVR  Floating-point overflow register

M  Modifier

This abbreviation usually appears in parentheses after the operand to signify that the latter may be modified, as in

LDX X N(M)

where N is the operand.

N  Operand

The exact composition of the operand for each operation statement is detailed in the description of the latter.

n  Indicates character position

The four 6-bit character positions within a word are numbered n0, n1, n2 and n3, starting from the most significant end of the word.

P  Peripheral interrupt indicator

This indicator is used with programs having one or more members, to note a request for activation by a flag-setting peripheral. Another indicator, M, is also used by program members in a similar way, to note a request for activities by another member.

V  Overflow register

The function and operation of this register is described on page 5 of Chapter 1.

X  Accumulator

This refers to any of locations 0 to 7 in a program. The X-field of an operation statement is its accumulator field.

X*  The succeeding accumulator

This refers to the accumulator that succeeds the specified accumulator. (It should be noted that X0 succeeds X7.)

Xc'm  A two-word counter-modifier

This refers to two accumulators, which need not be adjacent. The abbreviation is used only in the description of PLAN 4 macro-instructions; the two accumulators would normally be expected to contain a count and a modifier respectively.

DESCRIPTIONS OF OPERATION STATEMENTS

Each PLAN operation statement is dealt with in its own subsection below. The statements are in alphabetical order and the description of each starts on a fresh page.
PLAN 1,2,3

Function Add into accumulator X the value quoted in the operand field.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>ADN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the value to be added into X.

Execution
The value of N(M) is added into X. If C was left set by the previous instruction, 1 will be added to the
result in X. The contents of N(M) are unaltered.

If the operand is the symbolic name of a location previously defined under a #LOWER directive, then
the address of that location will be added into X. If the operand is written as a symbolic identifier
previously set by #DEFINE or #SET directives, the value that was assigned to that identifier is added
into X.

C is used and will be left clear.

V will be set if overflow occurs.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are
taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the
operand.

Machine Code Listing
101 X M N

Notes
1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to
the symbol (not PLAN 1).
ADNC

Add Direct Operand into X with Carry

Function  Add into accumulator X the value quoted in N(M), setting C if carry occurs.

Format  
Operation Code  ADNC
Accumulator  X
Operand  N(M)

N may be:
1  A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2  An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the value to be added into X.

Execution

The value of N(M) is added into X. If C was left set by the previous instruction, 1 will be added to the result in X. The contents of N(M) are unaltered. At the conclusion of this instruction, B0 of the result will always be zero.

If the operand is the symbolic name of a location previously defined under a #LOWER directive, then the address of that location will be added into X. If the operand is written as a symbolic identifier previously set by #DEFINE or #SET directives, the value that was assigned to that identifier is added into X.

C is used and will be set if a carry occurs. See Chapter 1, page 6.

V is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

In the following example, 437 is to be added to a triple-length integer held in X4, X5 and X6.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>20</th>
<th>28</th>
<th>22</th>
<th>16</th>
<th>8</th>
<th>0</th>
<th>OPERAND</th>
<th>48</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>0</th>
<th>MAYS ETVY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADNC</td>
<td>6</td>
<td>437</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETVY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MAYS ETVY</td>
</tr>
<tr>
<td></td>
<td>ADNC</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETVY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MAYS ETVY</td>
</tr>
<tr>
<td></td>
<td>ADNC</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ETVY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MAYS ETVY</td>
</tr>
</tbody>
</table>

Machine Code Listing

105  X  M  N

Notes

1  ADNC should only be used on the less significant words of multi-length operands.
2  If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
Add X to Store

Function
Add the contents of X to the contents of N(M).
Add the contents of XX* to the contents of N(M) and N + 1(M).

Format
Operation Code: ADS
Accumulator: X or XX*
Operand: N(M)

If X is specified, N may be:
1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

If X is specified, the operand N(M) refers to the location into which the contents of X are to be added.
If XX* is specified, the operand N(M) should be in the range 0 to 4094 and refers to the first of two consecutive locations into which the contents of XX* are to be added.

Execution
If X is specified in the accumulator field, the contents of X are added to the contents of N(M). If C is left set by the previous instruction, 1 will be added to the result in N(M). The contents of X are unaltered.
If XX* is specified, the contents of XX* are added to N(M) and N + 1(M). If C is left set by the previous instruction, 1 is added to the result in N(M) and N + 1(M). The contents of XX* are unaltered.

C is used and will be left clear.
V will be set if overflow occurs.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing
When X is specified: 011 X M N
When XX* is specified: 015 X* M N + 1
011 X M N

Notes
1 If X is specified and operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
   If XX* is specified and operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data (the symbol is given the value of the first location). Not PLAN 1.
ASCII
**ADX**

**Add into X**

*Function*  
Add the contents of N(M) to the contents of X.  
Add the contents of N(M) and N + 1(M) to the contents of XX*.

*Format*  

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>ADX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X or XX*</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

If X is specified, N may be:

1. A relative expression which refers to a lower data location, e.g.  
   (a) A symbolic name referring to a lower data location.  
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.  
   (a) A decimal integer in the range 0 to 4095.  
   (b) An octal integer in the range #0 to #7777.  
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).  
The operand N(M) refers to a location the contents of which are to be added to X.

If XX* is specified, N may be types 1 or 2 above. When N is type 2 it should be in the range 0 to 4094. The operand N(M) refers to the first of two consecutive locations the contents of which are to be added into XX*.

**Execution**

If X is specified, the contents of N(M) are added to the contents of X; if C was left set by the previous instruction, 1 is added to the result in X. The contents of N(M) are unaltered.

If XX* is specified the contents of N(M) and N + 1(M) are added to XX*. If C was left set by the previous instruction, 1 is added to the result in XX*. The contents of N + 1(M) are unaltered.

C is used and will be left clear.

V will be set if overflow occurs.

**Modification**  
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Machine Code Listing**

When X is specified:  
001 X M N

When XX* is specified:  
005 X* M N + 1  
001 X M N
Notes

1. If X is specified and operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

   If XX is specified and operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data (the symbol is given the value of the first location). Not PLAN 1.

2. ADX  XX  N(M)
   is equivalent to:
   ADXC  X  N + 1(M)
   ADX  X  N(M)
ABXC
ALLENT

Allot Peripheral

Function
Request allocation of a peripheral during the running of a program; or request information about a peripheral.

Format

Operation Code
ALLOT

The rest of the instruction can take three possible forms:

1. Accumulator
Blank

Operand
The symbolic name of the relevant peripheral (type and program's unit number, thus: LP0, TP2, etc.). The symbols for the permitted peripheral types are specified in Note 1.

2. Accumulator

Operand
The program's unit number of the relevant peripheral.

N(M), where N(M) is the type number of the peripheral. The type numbers of the permitted peripherals are specified in Note 1.

3. Accumulator

Operand
X, where X is an accumulator in which:

Bits 0 to 8 set to zero.

Bits 9 to 17 set to zero, unless used to specify peripheral property codes (see Note 5).

Bits 18 to 23 contain the program's unit number of the relevant peripheral.

N(M), where N(M) = 256 + the type number of the peripheral as specified for 2 above.

Operands of types 2 or 3 may have 512 added. See under 'Execution'. PLAN 1 compilers will not accept operands of type 1.

Execution

1. Basic Peripherals, the Interrogating Typewriter Facility, the Multiplexor and the Digital Incremental Plotter.

The ALLOT instruction requests allocation of a peripheral during the running of a program. This allows for more efficient use of peripherals as they need not be allotted to a program until they are required. When used with suitable Executives, this instruction can also specify properties required of the peripheral (except for the interrogating typewriter facility and the multiplexor).

Reply information to the ALLOT instruction is set into word 9 of the program, as described in Note 6.

If 512 is added to an operand of types 2 or 3, then the assigning of the peripheral is inhibited, but reply information is still set into word 9. This provides a means of ascertaining whether a basic peripheral is assigned to a program without the risk of causing an assignment.

2. Other Peripherals

Peripherals other than those specified in 1 above cannot be allotted by the ALLOT instruction, but the instruction will nevertheless set reply information into word 9 of the program, as described in Note 6. With magnetic tapes and cassette tapes the ALLOT instruction can therefore be used to ascertain whether a particular unit has been assigned to a program, and if so, its absolute unit number and (with suitable Executives) its properties. (Note that the ALLOT instruction cannot serve this purpose on a machine without a console typewriter, as on such machines all peripherals are assigned to the program: an affirmative reply would therefore be given whether a tape file had been opened or not.) With direct access devices the ALLOT instruction can be used to ascertain whether a particular file number is in use by a program.

C is not used and is left clear.
V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In extended data mode, the least significant 22 bits of N + M are taken as the operand. See Note 3 below.
Examples

Four forms that the ALLOT instruction can take are shown below. In each case, a paper tape punch, TP4, is allotted.

| LABEL | OPERATION | ACC | X | Y | Z | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |
|-------|-----------|-----|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ALLOUT 1 | TP4 |     |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| ALLOUT 2 | A |     |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| ALLOUT 7 | 256 |     |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| ALLOUT 7 | 256 |     |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Machine Code Listing

156 X M N

Notes

1 The peripheral types are specified in the operand field in the following ways:

Operand Type: 1 2 3

Basic Peripherals:
- Paper Tape Reader TR 0 256
- Paper Tape Punch TP 1 257
- Line Printer LP 2 258
- Card Reader CR 3 259
- Card Punch CP 4 260

Other Peripherals:
- Magnetic Tape MT 5 261
- Exchangeable Disc ED 6 262
- Magnetic Card File MC 7 263
- Magnetic Drum DR 9 265
- Interrogating Typewriter Facility or other single channel communicating device IT 10 266
- Multiplexor MX 11 267
- Cassette Tape CT 12 268
- Fixed Disc FD 13 269
- Alphanumeric Visual Display Unit 14 270
- Digital Incremental Plotter GP 20 276

2 Allotting of a peripheral of a given program's unit number does not cause all lower unit numbers to be allotted, as would the directive #PERIPHERAL.

3 The operand is modifiable, except in the case of a type 1 operand. If the operand is of type 2 or 3, the unit type may be modified. Note that a type 2 operand would be interpreted as a type 3 operand if B15 was set in the modifier.

4 The maximum number of peripheral units of any one type that can be allotted to a program is 16. The valid range of the program’s unit number is from 0 to 15 inclusive.

5 With dual program and multiprogram Executives it is possible to specify the properties required of the peripheral to be allotted. To take advantage of this facility it is necessary to use operands of type 3. Bits 9 to 17 of the accumulator, X, will then be used thus:

Bit 9 Set to zero if the relevant peripheral is required to have at least all the properties specified in bits 10 to 17.
Set to one if the relevant peripheral is required to have none of the properties specified in bits 10 to 17.

Bits 10 to 17 Contain the property code, which defines the properties of the relevant peripheral.

If the program is run with an Executive on which the properties system has not been implemented, or if a unit of the type and program's unit number requested by the ALLOT instruction has already been assigned to the program, bits 9 to 17 are ignored by Executive.

If bits 9 to 17 of the accumulator are set to zero, then no properties are specified for the relevant peripheral, and Executive will assign the first available unit of the requested type that it locates.

With peripherals other than direct access devices, the reply information set into word 9 as a result of an ALLOT instruction is as follows:

(a) if the program did not already have a peripheral of the type and program's unit number requested, and it is not possible to assign one by this ALLOT instruction, then word 9 is set negative, thus:

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>Set to 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 1 to 23</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

(b) if the program already had a peripheral of the type and program's unit number requested, or if a suitable unit has now been assigned by this ALLOT instruction, then:

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>Set to 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 1 to 8</td>
<td>Undefined</td>
</tr>
<tr>
<td>Bit 9</td>
<td>Set to 0 if the property system has not been implemented on the Executive concerned.</td>
</tr>
<tr>
<td></td>
<td>Set to 1 if the property system is implemented on the Executive concerned.</td>
</tr>
<tr>
<td>Bits 10 to 17</td>
<td>If bit 9 is 0: undefined.</td>
</tr>
<tr>
<td></td>
<td>If bit 9 is 1: contain the property code of the relevant peripheral.</td>
</tr>
<tr>
<td>Bits 18 to 23</td>
<td>Contain the absolute number of the relevant peripheral.</td>
</tr>
</tbody>
</table>

With direct access devices the reply information set into word 9 as a result of an ALLOT instruction is as follows:

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>Set to 0 if the specified file number is in use.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set to 1 if the specified file number is not in use.</td>
</tr>
<tr>
<td>Bits 1 to 23</td>
<td>Undefined.</td>
</tr>
</tbody>
</table>

If a program is run under the control of a GEORGE operating system, then:

(a) for on-line peripherals the use of property codes in an ALLOT instruction will not enable a peripheral of specified properties to be assigned. However, on completion of an ALLOT instruction the reply information as defined in Note 6 is set into word 9.

(b) for off-line peripherals, a zero reply is given to an ALLOT instruction. If a program relies on property codes or absolute device numbers in the reply to an ALLOT instruction, therefore, the user should ensure that the relevant peripheral is run on-line by the operating system.

Details of the property codes of particular devices may be found in the following 1990 Series manuals:

- Basic Peripherals.
- Magnetic Tape.
- Graph Plotter.
ANDN

Logical AND Direct Operand into X

Function
Perform a 'logical AND' on the contents of X using the value of N(M).

Format
- Operation Code: ANDN
- Accumulator: X
- Operand: N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.
   a. A symbolic name referring to a lower data location.
   b. A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   a. A decimal integer in the range 0 to 4095.
   b. An octal integer in the range #0 to #7777.
   c. A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the value to be used in the 'logical AND' operation.

Execution
A 'logical AND' operation is performed with the contents of X and the value of N(M), both words being regarded as 24-bit patterns rather than as numeric quantities. A 1-bit is placed in each bit position of X where both X and N(M) have a 1-bit; all other bits in X are set to 0. The contents of N(M) are unaltered.

If the operand is the symbolic name of a location previously defined under a #LOWER directive, the address of that location will be used as the appropriate bit pattern. If the operand is written as a symbolic identifier previously set by #DEFINE or #SET directives, the value that was assigned to that identifier is used as the appropriate bit pattern.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example
This example would cause bits 0 to 11 and bits 18 to 23 of X3 to be zeroized and bits 12 to 17 to be left unchanged.

Original contents of X3: 111 001 101 110 101 010 100 110

Original contents of operand: 000 000 000 000 111 111 000 000

| LABEL | OPERATION | ACC. | N | M | #7777 | OPERAND | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | PAGE |
|-------|-----------|------|---|---|-------|----------|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|-----|
| ANDN  | 3         | #7777|   |   |       |          |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |     |

Result in X3 (operand is unaltered): 000 000 000 000 101 010 000 000

Machine Code Listing
120 X M N

Notes
1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
ANNS

Logical AND into Store

Function  Perform a 'logical AND' on the contents of N(M) using the contents of X.

Format  

Operation Code  ANDS

Accumulator  X

Operand  N(M)

N may be:

1  A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.

2  An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the location on the contents of which the
'logical AND' is to be performed.

Execution

A 'logical AND' operation is performed with the contents of N(M) and the contents of X, both words
being regarded as 24-bit patterns rather than as numeric quantities. A 1-bit is placed in each bit
position of N(M) where both N(M) and X have a 1-bit; all other bits of N(M) are set to 0. The contents
of X are unaltered.

C  is not used and will be left clear.

V  is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are
taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

Original contents of SLOE:  010 111 101 001 110 111 000 011

Original contents of X3:  111 001 000 110 101 010 100 110

Result in SLOE:  010 001 000 000 100 010 000 010

(X3 is unaltered.)

Machine Code Listing

030  X M N

Notes

1  If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the
symbol (not PLAN 1).
**ANDX**

**Logical AND into X**

**Function**
Perform a 'logical AND' on the contents of X and the contents of N(M), into X.

**Format**

- **Operation Code**: ANDX
- **Accumulator**: X
- **Operand**: N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to the location the contents of which are to be used in the 'logical AND' operation.

**Execution**

A 'logical AND' operation is performed with the contents of X and the contents of N(M), both words being regarded as 24-bit patterns rather than as numeric quantities. A 1-bit is placed in each bit position of X where both X and N(M) have a 1-bit; all other bits of X are set to 0. The contents of N(M) are unaltered.

C is not used and will be left clear.

V is not used and remains unchanged.

**Modification**
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Example**

Original contents of X2: 111 010 101 110 011 111 000 110

Original contents of MASK: 101 110 100 000 111 101 111 101

| LABEL | OPERATION | ACC | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |
|       | ANDX      | 2   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0 |

Result in X2: 101 010 100 000 011 101 000 100

(MASK is unchanged.)

In this example MASK could have been defined under a #LOWER directive as #56407575. In PLAN 2 and 3, the instruction ANDX 2 '#56407575' would have achieved the same effect.
Machine Code Listing

020 X M N

Notes

1 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN I).
PLAN 3.4

Function   Activate program member X.
Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>AUTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>N(M)</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

For first activation of the member, the operand N(M) refers to the location that contains the instruction at which the member is to be entered; N(M) must not be zero. For subsequent activations N(M) must be zero.

Execution

If a program consists of more than one member, then, on loading, member 0 will be active, and all other members will be suspended awaiting activation by an AUTO statement with an operand which must not be zero. Any member other than member 0 may be activated by an AUTO statement in this form, but no one member may be activated by such a statement more than once.

For subsequent activations of any member, the form of AUTO statement with the operand equal to zero should be used and this will cause the member to be entered at the instruction following the SUSAR or SUSIN by which the member was last suspended. If the member X is active at the time the AUTO statement is issued, then its indicator, M, will be set and will remain set until member X itself issues a SUSIN statement which will clear M, but not suspend member X.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing

163  X  M  N

Notes

1. The statement is illegal if X specifies either the current member or a non-existent member.

2. On 1901 processors and 1902 and 1903 single-program processors a program may not have more than one member.
BCC

Branch if C is Clear

Function Cause the program to branch to a specified location if the C register is clear.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>BCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. A relative operand.

The operand N refers to the program location to which control is transferred if C is clear.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 3 below).

Execution

The BCC instruction tests the state of the Carry register (C). If this register is clear the program branches to the location specified by N. If C is not clear the program continues to the next instruction.

C is used (see above) and will be left clear.

V is not used and remains unchanged.

Modification This instruction has no M-field.

Example

In this example, the contents of locations VERA and VERB are compared; if they are equal a branch is made to an instruction labelled NEXT.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>ACC</th>
<th>0</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDX</td>
<td>A</td>
<td>VERA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TXU</td>
<td>A</td>
<td>VERB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BCC</td>
<td></td>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Machine Code Listing

074 6 N
Notes

1 Branches dependent on the state of the Carry register are primarily intended for use with the comparison functions, TXU and TXL. Generally a comparison function must be followed immediately by a BCC or BCS instruction, otherwise the setting of C may be lost.

2 The compiler inserts 6 in the accumulator field in the machine-code instruction generated to distinguish it from other 074 instructions.

3 In PLAN 4, the BCC instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BCC instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BCHX

Branch on Character Indexing

(The PLAN 4 macro-instruction BCHX is described separately.)

Function
Adjust character index word and branch if the count is non-zero.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>BCHX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
</tbody>
</table>

Any accumulator (X0 to X7) may be used, but as only X1, X2 and X3 are available for modification purposes, these are normally used.

Operand
N

N may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand N refers to the program location to which control is transferred if the count in the character index word is not reduced to zero by the instruction.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 3 below).

Execution
The BCHX instruction operates on a character index word (which is described in Chapter 1, page 3). The character index word is held in the specified accumulator and is notionally divided into three parts, bits 0 to 1, bits 2 to 8 and bits 9 to 23. In bits 2 to 8 there is a count. The BCHX instruction subtracts 1 from bits 2 to 8 (the count) and adds 1 to bits 0 to 1. When bits 0 to 1 contain 3 and a further 1 is added, then there is a circular carry so that bits 0 to 1 become zero and 1 is added to bits 9 to 23. The BCHX causes a branch to the address specified in N if the count in bits 2 to 8 is not zero after 1 has been subtracted from it. If, however, the count has become zero then the next operation in sequence is obeyed. The maximum count which may be held in bits 2 to 8 is 127. If the count is initially zero this will be effectively equivalent to a count of 126.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
The BCHX statement has no M-field.

Execution in Extended Data Mode
The BCHX instruction treats the word in X as consisting only of two parts, bits 0 to 1 and bits 2 to 23. 1 is added to bits 0 to 1, with a carry into bits 2 to 23. No count operates; the instruction branches unconditionally to the location specified by N (which will usually contain a BCT instruction).

Machine Code Listing
064 X N
Notes

1. This operation is designed primarily for use when handling six-bit characters. For character working the index word will also be used as a modifier by the operations LDCH and DCH. When used in this way bits 0 to 1 of the index word constitute a character pointer, pointing successively to characters n0, n1, n2, n3 within a word, while bits 9 to 23 operate as a modifier which determines the addresses of successive words. The sequence will run: n0, n1, n2, n3, n+1.0, n+1.1, n+1.2, n+1.3, and so on. In extended data mode all of bits 2 to 23 are used as a word modifier.

2. The count required in an index word may be set by a LDCT operation.

3. In PLAN 4, the BCHX instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BCHX instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BCHX

Branch on Character Indexing

Function  Adjust character modifier and count, and branch if the count is non-zero.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>BCHX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>$X_c X_m$</td>
</tr>
</tbody>
</table>

Any accumulators (X0 to X7) may be used, but as only X1, X2 and X3 are available for modification purposes, these are normally used for $X_m$.

$X_c$ and $X_m$ need not be adjacent accumulators.

| Operand | N            |

N may be:

1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. A relative operand,

The operand N refers to the program location to which control is transferred if the least significant 15 bits of $X_c$ are not equal to zero.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 4 below).

Execution

The BCHX macro-instruction will generate two machine code instructions. It is intended to be used in extended data mode, to provide a similar result to that obtained from the use of a BCHX instruction in compact mode. The character modifier in $X_m$ is increased by one, the count in $X_c$ is decreased by one, and a branch to N takes place unless the count is reduced to zero.

$C$ is not used and will be left clear.

$V$ is not used and remains unchanged.

Modification  The BCHX macro-instruction has no M-field.

Machine Code Listing

<table>
<thead>
<tr>
<th>Code</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>064</td>
<td>$X_m *+1$</td>
</tr>
<tr>
<td>066</td>
<td>$X_c$</td>
</tr>
</tbody>
</table>

Notes

1. BCHX  $X_c X_m$  N
   is equivalent to
   BCHX  $X_m *+1$
   BCT  $X_c$  N

Chapter 4
2 The BCT instruction generated by this macro-instruction is available only on processors with extended data mode facilities.

3 The exact manner in which the contents of \( X_m \) are incremented depends on the address mode in which the program is operating; see the description of the BCHX instruction.

4 In PLAN 4, the BCT instruction generated by this macro-instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BCT instruction generated by this macro-instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BS

Branch if C is Set

Function

Cause the program to branch to a specified location if the C register is set.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>BCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed
       decimal or octal integer.

2. A relative operand.

The operand N refers to the program location to which control is transferred if C is set.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 3 below).

Execution

The BCS instruction tests the state of the Carry register (C). If this register is set the program branches to the location specified by N. If C is not set the program continues to the next instruction.

C is used (see above) and will be left clear.

V is not used and remains unchanged.

Modification

The BCS statement has no M-field.

Example

In this example, it is required to compare the contents of location LOLA and LIAM. If their contents are not equal the program must branch to an instruction labelled HOME.

| LABEL | OPERATION | ACC | 16 | 20 | 24 | 26 | 28 | 30 | 32 | 34 | 36 | 38 | 40 | 42 | 44 | 46 | 48 | 50 | 52 | 54 | 56 | 58 | 60 | 62 | 64 | 66 | 68 |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | LDX       | A   | LOLA          |
|       | TXU       | A   | LIAM          |
|       | BCS       |     | HOME          |

Machine Code Listing

074 5 N

Notes

1. Branches dependent of the state of the Carry register are primarily intended for use with the comparison functions, TXU and TXL. Generally a comparison function must be followed immediately by a BCC or BCS instruction otherwise the setting of C will be lost.

2. The compiler inserts 5 in the accumulator field in the machine-code instruction generated to distinguish it from other 074 instructions.
In PLAN 4, the BCS instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BCS instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BCT

Branch on Count
(Available on 1902A, 1903A and Processors with Extended Data Mode Facilities)

Function
Cause a branch to the location specified in N if the count in X does not equal zero.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>BCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N</td>
</tr>
</tbody>
</table>

N may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. A relative operand.

The operand N refers to the program location to which control is transferred if the modifier part of X is not equal to zero.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 1 below).

Execution
The BCT instruction subtracts 1 from the least significant 15 bits of the specified accumulator. It then transfers control to the program location specified by N, unless B9 to B23 of X has been reduced to zero, in which case control passes to the next instruction in sequence. The most significant nine bits of X are unchanged.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
The BCT statement has no M-field.

Execution in Extended Data Mode
The BCT instruction subtracts 1 from the least significant 22 bits of the specified accumulator. It then transfers control to the program location specified by N, unless B2 to B23 of X has been reduced to zero, in which case control passes to the next instruction in sequence. The most significant two bits of X are unchanged.

Example
In this example, it is required to clear 1250 locations, from CARP to CARP + 1249.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STOZ</td>
<td>CARP-1</td>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BCT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is assumed in this example that CARP is not the first location of its area; if it were, a compilation error would arise on the STOZ instruction (see page 16 of Chapter 2).
Machine Code Listing

066 X N

Notes

1 In PLAN 4, the BCT instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BCT instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BDX

Branch on Double Indexing
(The PLAN 4 macro-instruction BDX is described separately.)

Function
The modifier in bits 9 to 23 of X is increased by 2 and the count in bits 0 to 8 is decreased by 1. If the count in X is not equal to zero, branch to the location specified in N.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Accumulator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDX</td>
<td>X</td>
<td>N</td>
</tr>
</tbody>
</table>

N may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand N refers to the program location to which control is transferred if the count in X is not equal to zero.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 1 below).

Execution
This instruction treats the contents of the accumulator X as an index word, adding 2 to bits 9 to 23, subtracting 1 from bits 0 to 8, and then branching to N unless the new value of the count is zero. There is no carry between bits 9 to 23 and bits 0 to 8 (or vice versa). The maximum value of the modifier is 32, 767 and of the count 511. An initial count of zero is effectively equivalent to 512.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
The BDX statement has no M-field.

Execution in Extended Data Mode
The BDX instruction treats accumulator X as containing the modifier in the least significant 22 bits. 2 is added to bits 2 to 23. No count operates; the instruction branches unconditionally to N (which will usually contain a BCT instruction).

Example
In this example it is required to find the sum of eight double-length numbers contained in locations WASTE to WASTE + 15; the result is to be placed in RES. If overflow occurs a branch is made to ERROR.
The BDX instruction decreases the count (in B0 to B8) in X2 by 1 and adds 2 to the modifier (in B9 to B23) to obtain the address of the next double-length word to be added. If the count is zero the program proceeds to the next instruction and places the sum in RES and RES + 1. If the count is not zero the program branches to the location specified in the operand field.

Note that the double-length instructions ADX and STO, used above, are macro-instructions available only in PLAN 3 or 4. If PLAN 1 or 2 is being used these instructions would have to be replaced by the corresponding single-length instructions.

Machine Code Listing

062 X N

Notes

1. In PLAN 4, the BDX instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BDX instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BDX

Branch on Double Indexing

Function
Increase the modifier in \( X_m \) by 2 and decrease the count in \( X_c \) by 1. If the count is not equal to zero, branch to the location specified by \( N \).

Format

\begin{align*}
\text{Operation Code} & : \text{BDX} \\
\text{Accumulator} & : X_cX_m \\
\text{Operand} & : N
\end{align*}

\( X_c \) and \( X_m \) need not be adjacent accumulators.

N may be:

1. A relative expression which refers to a program instruction, e.g.
   a. A symbolic name that is the label of a program instruction.
   b. A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. A relative operand.

The operand \( N \) refers to the program location to which control is transferred if the least significant 15 bits of \( X_c \) are not equal to zero.

In PLAN 4, \( N \) may be preceded by a colon as the first character of the operand field (see Note 3 below).

Execution

The BCHX macro-instruction will generate two machine code instructions. It is intended to be used in extended data mode, to provide a similar result to that obtained from the use of a BDX instruction in compact mode. The modifier in \( X_m \) is increased by 2, the count in \( X_c \) is decreased by 1, and a branch to \( N \) takes place unless the count is reduced to zero. A further 1 is added to \( X_m \) if \( C \) was left set by the previous instruction.

\( C \) is used and will be left clear.

\( V \) will be set if overflow occurs.

Modification
The BDX macro-instruction has no M-field.

Machine Code Listing

\begin{align*}
101 & \text{ XM 0 2} \\
066 & \text{ Xc N}
\end{align*}

Notes

1. BDX \( X_cX_m \) \( N \) is equivalent to
   \begin{align*}
   \text{ADN} & \text{ XM 2} \\
   \text{BCT} & \text{ Xc N}
   \end{align*}

2. The BCT instruction generated by this macro-instruction is available only on processors with extended data mode facilities.
In PLAN 4, the BCT instruction generated by this macro-instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BCT instruction generated by this macro-instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BFP

Branch on State of Floating-point Accumulator or Floating-point Overflow

(Available on 1902A, 1903A and Processors with Extended Data Mode Facilities)

Function
Test the state of the Floating-point Accumulator or Floating-point Overflow and branch on the condition specified by the X address of the operation.

Format
Operation Code  BFP
Accumulator  X

The X part of the operation does not refer to an accumulator, but must be set to a value in the range 0 to 5; see Execution.

Operand
N

N may be:
1  A relative expression which refers to a program instruction, e.g.
   (a)  A symbolic name that is the label of a program instruction.
   (b)  A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2  A relative operand.

The operand N refers to the program location to which control is transferred if the condition specified by X is satisfied.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 1 below).

Execution
The value in the accumulator field determines the condition on which branching is to occur as follows:

X = 0  Branch to N if the content of A = 0
X = 1  Branch to N if the content of A ≠ 0
X = 2  Branch to N if the content of A > 0 (i.e. sign bit of argument = 0)
X = 3  Branch to N if the content of A < 0 (i.e. sign bit of argument = 1)
X = 4  Branch to N if FOVR is clear
X = 5  Branch to N if FOVR is set.

The contents of A will remain unchanged. The BFP instruction is not carried out until any current process in A is completed.

FOVR is used (see above) but remains unaltered.

C is not used and will be left clear.

V is not used and remains unchanged if X = 4 or 5. If X = 0, 1, 2 or 3, V will be set if FOVR is set.

Modification
The BFP statement has no M-field.

Machine Code Listing
076  X  N
Notes

1 In PLAN 4, the BFP instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BFP instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
Branch if X is Negative

Function
Test the contents of X and if they are negative transfer control to the location specified in N.

Format

- **Operation Code**: BNG
- **Accumulator**: X
- **Operand**: N

N may be:

1. A relative expression which refers to a program instruction, e.g.
   - (a) A symbolic name that is the label of a program instruction.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand N refers to the program location to which control is transferred if X is negative.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 1 below).

Execution

The BNG instruction tests the contents of X, and if it is negative (i.e. B0 = 1) causes a branch to the location specified by N. If X is not negative the program continues to the next instruction. The contents of X will remain unchanged.

- **C**: is not used and will be left clear.
- **V**: is not used and remains unchanged.

Modification

The BNG statement has no M-field.

Example

A number in the word LOLA is to be subtracted from a number in X4. If the result in X4 is negative a branch is made to ERROR, otherwise the answer is stored in RES.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>X4</th>
<th>X3</th>
<th>X2</th>
<th>X1</th>
<th>X0</th>
<th>OPERAND</th>
<th>IN</th>
<th>OUT</th>
<th>N0</th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S,BX</td>
<td>A LOLA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNG</td>
<td>A ERROR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S,T0</td>
<td>A RES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Machine Code Listing

056 X N

Notes

1. In PLAN 4, the BNG instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BNG instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BNZ

Branch if X is Not Zero

PLAN 1.2.3.4

Function
Test the contents of X and if they are not zero transfer control to the location specified in N.

Format
- Operation Code: BNZ
- Accumulator: X
- Operand: N

N may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand N refers to the program location to which control is transferred if X is not equal to zero.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 1 below).

Execution
The BNZ instruction tests the content of X, and if it is not zero causes a branch to the location specified by N. If X is zero the program continues to the next instruction. The contents of X will remain unchanged.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
The BNZ statement has no M-field.

Machine Code Listing
052 X N

Notes
1. In PLAN 4, the BNZ instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BNZ instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BPZ

Branch if X is Positive or Zero

Function  Test the contents of X and if they are positive or zero transfer control to the location specified in N.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Accumulator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPZ</td>
<td>X</td>
<td>N</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. A relative operand.

The operand N refers to the program location to which control is transferred if X is positive or zero.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 1 below).

Execution

The BPZ instruction tests the content of X, and if it is positive or zero (i.e. B0 = 0) causes a branch to the location specified by N. If X is not positive or zero (i.e. if it is negative) the program continues to the next instruction. The contents of X remain unchanged.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification  The BPZ statement has no M-field.

Machine Code Listing

054  X  N

Notes

1. In PLAN 4, the BPZ instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BPZ instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
Unconditional Branch

Function  Transfer control unconditionally to the location specified by the operand.

Format  

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Accumulator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRN</td>
<td>Blank</td>
<td>N</td>
</tr>
</tbody>
</table>

N may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand N refers to the program location to which control is transferred.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 3 below).

Execution

The BRN instruction results in an unconditional branch to the location specified by N.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification  The BRN statement has no M-field.

Machine Code Listing

074 0 N

Notes

1. The BRN statement is frequently used to terminate small sections of program and transfer control to other sections.

2. The compiler inserts 0 in the accumulator field in the machine-code instruction generated to distinguish it from other 074 instructions.

3. In PLAN 4, the BRN instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules as stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BRN instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BSP

Backspace

Function  Backspace a magnetic tape one block.

Format  
  Operation Code  BSP
  Accumulator  A decimal digit in the range 0 to 7, specifying the program's unit number for the magnetic tape unit.
  Operand  Blank

Execution  
BSP causes the magnetic tape on the specified unit to be backspaced one block, so that the block may be read again.

Modification  The BSP macro-instruction has no M-field.

Machine Code Listing

157 X 0 n LT

where n is the relative address in the literal table of the first of two consecutive locations which hold a control area of form:

  first word:  5/453
  second word: 0

Notes  
1  This is one of a set of magnetic tape macro-instructions which permit the execution of simple basic functions on magnetic tape without requiring the programmer to set up control areas. (See also: BTM, CLOSE, FTM, REW, SCR, UNL, WTM.)
PLAN 3.4
(Magnetic Tape Macro-instruction)

Function
Rewinds a magnetic tape to the previous tape mark.

Format
Operation Code: BTM
Accumulator: A decimal digit in the range 0 to 7, specifying the program's unit number for the magnetic tape unit.
Operand: Blank

Execution
BTM causes the tape on the specified unit to be rewound until a tape mark is detected. The tape will stop positioned to read the tape mark again.

Modification
The BTM macro-instruction has no M-field.

Machine Code Listing
157 X 0 n LT
where n is the relative address in the literal table of the first of two consecutive locations which hold a control area of form:

first word: 5/6
second word: 0

Notes
1 This is one of a set of magnetic tape macro-instructions which permit the execution of simple basic functions on magnetic tape without requiring the programmer to set up control areas. (See also: BSP, CLOSE, FTM, REW, SCR, UNL, WTM.)
BUX

Branch on Unit Indexing

(The PLAN 4 macro-instruction BUX is described separately.)

Function
The modifier in bits 9 to 23 of X is increased by 1 and the count in bits 0 to 8 is decreased by 1. If the count in X is not zero there is then a branch to the location specified in N.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>BUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. A relative operand.

The operand N refers to the program location to which control is transferred if the counter in X is not equal to zero.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 1 below).

Execution
This instruction treats the contents of the accumulator X as an index word, adding 1 to bits 9 to 23, subtracting 1 from bits 0 to 8, and then branching to N unless the new value of the count is zero. There is no carry between bits 9 to 23 and bits 0 to 8 (or vice versa). The maximum value of the modifier is 32,767 and of the counter 511. An initial count of zero is effectively equivalent to 512.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
The BUX statement has no M-field.

Execution in Extended Data Mode
The BUX instruction treats accumulator X as containing the modifier in the least significant 22 bits. 1 is added to bits 2 to 23. No count operates; the instruction branches unconditionally to N (which will usually contain a BCT instruction).

Examples

1. It is required to find the sum of the contents of eight words starting at location WASTE: the result is to be placed in RES. If overflow occurs a branch is to be made to ERROR.

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 2 | 6 | 6 | 0 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| 4 | 6 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| 6 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| 7 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| 8 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| 9 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| 10| . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| 11| . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |

42 Chapter 4
The BUX instruction decreases the count (in B0 to B8) in X3 by 1 and adds 1 to the modifier (in B9 to B23) to obtain the address of the next word to be added. If the count is zero the program proceeds to the next instruction and places the sum in RFS. If the count is not zero the program branches to the location specified in the operand field.

2. It is required to clear 250 locations, from LOCA to LOCA + 249 in upper data.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>X</th>
<th>N</th>
<th>V</th>
<th>H</th>
<th>W</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX</td>
<td>2</td>
<td>'250/LOCA'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LOAD INDEX NEXT WORD</td>
</tr>
<tr>
<td>STOP</td>
<td>0 (2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CLEAR NEXT WORD</td>
</tr>
<tr>
<td>BUX</td>
<td>2</td>
<td>N-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BRANCH IF COUNTER NOT ZERO</td>
</tr>
</tbody>
</table>

Machine Code Listing

060  X  N

Notes

1. In PLAN 4, the BUX instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BUX instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
Branch on Unit Indexing

(Macro-instruction)

Function: Increase the modifier in $X_m$ by 1 and decrease the count in $X_c$ by 1. If the count is not equal to zero, branch to the location specified by $N$.

Format:
- Operation Code: BUX
- Accumulator: $X_c X_m$
- $X_c$ and $X_m$ need not be adjacent accumulators.
- Operand: $N$

$N$ may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand $N$ refers to the program location to which control is transferred if the least significant 15 bits of $X_c$ are not equal to zero.

In PLAN 4, $N$ may be preceded by a colon as the first character of the operand field (see Note 3 below).

Execution:
The BUX macro-instruction will generate two machine code instructions. It is intended to be used in extended data mode, to provide a similar result to that obtained from the use of a BUX instruction in compact mode. The modifier in $X_m$ is increased by 1, the count in $X_c$ is decreased by 1, and a branch to $N$ takes place unless the count is reduced to zero. A further 1 is added to $X_m$ if $C$ was left set by the previous instruction.

$C$ is used and will be left clear.
$V$ will be set if overflow occurs.

Modification: The BUX macro-instruction has no M-field.

Machine Code Listing:

```
101 $X_m$ 0 1
066 $X_c$  N
```

Notes:
1. BUX $X_c X_m$ N
   is equivalent to
   ADN $X_m$ 1
   BCT $X_c$  N
2. The BCT instruction generated by this macro-instruction is available only on processors with extended data mode facilities.
3 In PLAN 4, the BCT instruction generated by this macro-instruction is compiled as a direct branch, a relative branch, or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BCT instruction generated by this macro-instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
Branch if V Clear:

Function

Test the state of the Overflow register (V) and, if clear, transfer control to the location specified in N.

Format

Operation Code: BVC
Accumulator: Blank
Operand: N

N may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand N refers to the program location to which control is transferred if V is clear.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 2 below).

Execution

The BVC instruction tests the state of the Overflow register (V), and if it is clear causes a branch to the location specified by the operand. If V is not clear the program continues to the next instruction.

C is not used and will be left clear.

V is used (see above) and remains unchanged.

Modification The BVC statement has no M-field.

Machine Code Listing

074 3 N

Notes

1. The compiler inserts 3 in the accumulator field in the machine-code instruction to distinguish it from other 074 instructions.
2. In PLAN 4, the BVC instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BVC instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
PLAN 1.2.3.4

**Function**
Test the state of the Overflow register (V) and if clear transfer control to the location specified in N. V is inverted.

**Format**

- **Operation Code**: BVCI
- **Accumulator**: Blank
- **Operand**: N

N may be:

1. A relative expression which refers to a program instruction, e.g.
   a) A symbolic name that is the label of a program instruction.
   b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. A relative operand.

The operand N refers to the program location to which control is transferred if V is clear.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 2 below).

**Execution**
The BVCI instruction tests the state of the Overflow register (V), and if it is clear causes a branch to the location specified by N. If V is not clear the program continues to the next instruction.

- **C** is not used and will be left clear.
- **V** is used and will be inverted (i.e. if it is found clear it will be left set, and if it is found set it will be left clear).

**Modification**
The BVCI statement has no M-field.

**Machine Code Listing**

074 7 N

**Notes**

1. The compiler inserts 7 in the accumulator field in the machine-code instruction generated to distinguish it from other 074 instructions.

2. In PLAN 4, the BVCI instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BVCI instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
Branch if V Clear and Clear V

Function  Test the state of the Overflow register (V) and if clear transfer control to the location specified in N. V is cleared.

Format

| Operation Code | BVCR |
| Accumulator    | Blank |
| Operand        | N     |

N may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand N refers to the program location to which control is transferred if V is clear.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 2 below).

Execution

The BVCR instruction tests the state of the Overflow register (V), and if it is clear causes a branch to the location specified by N. If V is not clear the program continues to the next instruction.

C is not used and will be left clear.

V is used and will be left clear.

Modification  The BVCR statement has no M-field.

Machine Code Listing

074 4 N

Notes

1. The compiler inserts 4 in the accumulator field in the machine-code instruction generated to distinguish it from other 074 instructions.

2. In PLAN 4, the BVCR instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BVCR instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
Plan 1.2.3.4

Function  Test the state of the Overflow register (V) and, if set, transfer control to the location specified in N.

Format  

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>BVS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N</td>
</tr>
</tbody>
</table>

N may be:

1 A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2 A relative operand.

The operand N refers to the program location to which control is transferred if V is set.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 2 below).

Execution

The BVS instruction tests the state of the Overflow register (V), and if it is set causes a branch to the location specified by N. If V is not set the program continues to the next instruction.

C is not used and will be left clear.

V is used and remains unchanged.

Modification  The BVS statement has no M-field.

Machine Code Listing

074 1 N

Notes

1 The compiler inserts 1 in the accumulator field in the machine-code instruction generated to distinguish it from other 074 instructions.

2 In PLAN 4, the BVS instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BVS instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BVSR

Branch if V Set; Leave V Clear

Function
Test the state of the Overflow register (V) and, if set, transfer control to the location specified in N. V is left clear.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>BVSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N</td>
</tr>
</tbody>
</table>

N may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand N refers to the program location to which control is transferred if V is set.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 2 below).

Execution
The BVSR instruction tests the state of the Overflow register (V), and if it is set causes a branch to the location specified by N. If V is not set the program continues to the next instruction.

C is not used and will be left clear.
V is used and is left clear.

Modification
The BVSR statement has no M-field.

Machine Code Listing
074 2 N

Notes
1. The compiler inserts 2 in the accumulator field in the machine-code instruction generated to distinguish it from other 074 instructions.
2. In PLAN 4, the BVSR instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BVSR instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
Plan 3.4
(Macro-instruction)

Function
Compare the contents of X with the contents of $N_1(M)$ and if equal branch to $N_2$.
Compare the contents of $XX^*$ with the contents of $N_1(M)$ and $N_1 + 1(M)$ and if equal branch to $N_2$.

Format

Operation Code
BXE

Accumulator
X or $XX^*$

Operand
$N_1(M)$, $N_2$

If X is specified, $N_1$ may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.
3. A literal.

$N_2$ may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand refers to two locations. The first, $N_1(M)$, is the location whose contents are compared with the contents of $X$, and the second, $N_2$, is the program location to which control is transferred if the equality test holds.

If $XX^*$ is specified, $N_1$ may be types 1 and 2 given above for $N_1$ when $X$ is specified; $N_2$ is as above. The operand refers to three locations.

$N_1(M)$ is the first of two consecutive locations whose contents are compared with the contents of $XX^*$ and $N_2$ is (as before) the program location to which control is transferred if the equality test holds.

In PLAN 4, a colon may be inserted between the comma and $N_2$ (see Note 4 below).

Execution
In the case of single-length locations, $X$ is specified and the BXE macro-instruction will generate two machine-code instructions. Its effect is to compare the contents of $X$ with the contents of $N_1(M)$. If they are equal, and if $C$ was not set by the previous instruction, control is transferred to the point in the program specified by $N_2$. Otherwise the program continues with the next instruction in sequence. In the case of double-length locations, $XX^*$ is specified and the BXE macro-instruction will generate three machine-code instructions. Its effect is to compare the contents of $XX^*$ with the contents of $N_1(M)$ and $N_1 + 1(M)$. 

Chapter 4
If they are equal, and if C was not left set by the previous instruction, control is transferred to the point in the program specified by \( N_2 \).

If they are not equal the program continues with the next instruction in sequence.

C is used and will be left clear (see Note 1).

V is not used and remains unchanged.

**Modification** The BXE statement has an M-field. When modified, the least significant 15 bits of \( N_1 + M \) are taken as the operand. In the extended data mode, the least significant 22 bits of \( N_1 + M \) are taken the operand.

**Examples**

1. It is required to compare the contents of DESK and WASTE and if they are equal branch to SACK.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>X</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>X</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDX</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DESK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BXE</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WASTE, SACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   The macro-instruction will cause the following series of instructions to be generated in machine code:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>X</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>X</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TXU</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WASTE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BCC</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. It is required to compare the contents of a double-length number in CAB and CAB + 1 with a double-length number in FILE and FILE + 1. If they are equal branch to DUMP.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>X</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>X</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDX</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AB. CAB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BXE</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FILE, DUMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   The macro-instruction BXE will cause the following series of instructions to be generated in machine code:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>X</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>X</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TXU</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FILE+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BC. FILE</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DUMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   Machine Code Listing

   When X is specified:  
   
   026  X  M  \( N_1 \)  
   074  6  \( N_2 \)

   When XX* is specified:  
   
   026  X*  M  \( N_1 + 1 \)  
   026  X  M  \( N_1 \)  
   074  6  \( N_2 \)

   \( N_2 \) is the label to which control is transferred.

   **Notes**

   1 The Carry register (C) should be clear before the BXE macro-instruction is obeyed, except in testing quantities of more than double-length, where one or more TXU's must be used, and C may thus be used before reaching the BXE. See under TXU.

   52 Chapter 4
2 If X is specified and operand N₁, type 1(a), is an undefined symbol, the compiler will allocate a location in lower data to the symbol. If XX* is specified and operand N₁, type 1(a), is an undefined symbol, the compiler will allocate two consecutive locations in lower data (the symbol being given the value of the first location).

3 BXE X N₁(M), N₂

is equivalent to
TXU X N₁(M)
BCC N₂

BXE XX* N₁(M), N₂

is equivalent to
TXU X* N₁ + 1(M)
TXU X N₁(M)
BCC N₂

4 In PLAN 4, the BCC instruction generated by this macro-instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the N₂ operand denotes that the BCC instruction generated by this macro-instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
BXGE

Branch if X greater than or equal to N(M)

**Function**
Test the contents of X and if they are greater than or equal to the contents of N₁(M) branch to N₂.

Test the contents of XX* and if they are greater than or equal to the contents of N₁(M) and N₁ + 1(M) branch to N₂.

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>BXGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X or XX*</td>
</tr>
<tr>
<td>Operand</td>
<td>N₁(M), N₂</td>
</tr>
</tbody>
</table>

If X is specified, N₁ may be:

1. A relative expression which refers to a lower data location, e.g.
   - A symbolic name referring to a lower data location.
   - A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - A decimal integer in the range 0 to 4095.
   - An octal integer in the range #0 to #7777.
   - A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal.

N₂ may be:

1. A relative expression which refers to a program instruction, e.g.
   - A symbolic name that is the label of a program instruction.
   - A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. A relative operand.

The operand refers to two locations. The first, N₁(M), is the location whose contents are compared with the contents of X, and the second, N₂, is the program location to which control is transferred if the test conditions are satisfied.

If XX* is specified, N₁ may be types 1 and 2 given above for N₁ when X is specified; N₂ is as above. The operand refers to three locations, N₁(M) is the first of two consecutive locations whose contents are compared with the contents of XX* and N₂ is the program location to which control is transferred if the test conditions are satisfied.

In PLAN 4, a colon may be inserted between the comma and N₂ (see Note 5 below).

**Execution**

In the case of single-length locations, X is specified and the BXGE macro-instruction will generate two machine-code instructions. Its effect is to compare the contents of X with the contents of N₁(M). Both words are considered as 24-bit positive binary numbers and thus no special significance is attributed to the sign bit. If C is initially clear and the contents of X are greater than or equal to the contents of N₁(M), control is transferred to the point in the program specified by N₂. If C was left set by the previous instruction and the words being compared are equal, control is transferred to N₂. Otherwise the program continues with the next instruction in sequence.
In the case of double-length locations XX* is specified and the BXGE macro-instruction will generate three machine-code instructions. Its effect is to compare the contents of XX* with the contents of N₁(M) and N₁ + 1(M). Both pairs of words are considered as 48-bit positive binary numbers and thus no special significance is attached to the sign bit. If C is initially clear and the contents of XX* are greater than or equal to the contents of N₁(M) and N₁ + 1(M) control is transferred to the point in the program specified by N₂. If C was left set by the previous instruction and the words being compared are equal, control is transferred to N₂. Otherwise the program continues with the next instruction in sequence.

C is used and will be left clear (see Note 1).

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Examples
1 It is required to test whether the contents of COAT are greater than or equal to the contents of VEST. If they are, branch to WARDS.

```
0
1
LABEL  
LDX 6. COAT
BXGE 6. VEST, WARDS
```

The macro-instruction will cause the following series of instructions to be generated in machine code:

```
0
1
LABEL  
TAX 6. VEST
BCC WARDS  
```

If C is set, branch to WARDS;

```
0
1
LABEL  
TAX 6. FRANNT+1
BCC GLASS  
```

If C is set, branch to GLASS;

2 It is required to test whether a double-length number in ZOOEY and ZOOEY + 1 is less than a double-length number in FRANNT and FRANNT + 1. If it is not, branch to GLASS.

```
0
1
LABEL  
LDX 6. ZOOEY
BXGE 6. FRANNT, GLASS
```

The macro-instruction will cause the following series of instructions to be generated in machine code:

```
0
1
LABEL  
TAX 7. FRANNT+1
BCC GLASS  
```

If C is set, branch to GLASS.

```
0
1
LABEL  
TAX 7. FRANNT
BCC GLASS  
```

If C is set, branch to GLASS.

Machine Code Listing
When X is specified: 027 X M N₁
074 6 N₂

When XX* is specified: 027 X* M N₁ + 1
027 X M N₁
074 6 N₂

(N₂ is the label to which control is transferred.)
Notes

1. The Carry register (C) should be clear before the BXGE macro-instruction is obeyed, except in testing quantities of more than double-length, where one or more TXL's must be used, and C may thus be used before reaching the BXGE. See under TXL.

2. The fact that the two words compared are considered as 24- or 48-bit positive binary numbers makes it possible to use the BXGE statement with alphanumeric data. When using it to compare numbers there is no problem if the numbers have the same sign. If the numbers being compared have opposite signs it is necessary to invert the sign bits of both numbers before making the comparison.

3. If X is specified and operand \( N_1 \), type 1(a), is an undefined symbol, the compiler will allocate a location in lower data to the symbol.

   If \( XX^* \) is specified and operand \( N_1 \), type 1(a), is an undefined symbol, the compiler will allocate two consecutive locations in lower data (the symbol is given the value of the first location).

4. \( \text{BXGE } X \quad N_1(M), N_2 \)

   is equivalent to

   \( \text{TXL } X \quad N_1(M) \)

   \( \text{BCC } N_2 \)

   \( \text{BXGE } XX^* \quad N_1(M), N_2 \)

   is equivalent to

   \( \text{TXL } X^* \quad N_1 + 1(M) \)

   \( \text{TXL } X \quad N_1(M) \)

   \( \text{BCC } N_2 \)

5. In PLAN 4, the BCC instruction generated by this macro-instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the \( N_2 \) operand denotes that the BCC instruction generated by this macro-instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
PLAN 3.4

(Macro-instruction)

Function Test the contents of X and if they are less than the contents of \( N_1(M) \) branch to \( N_2 \).

Test the contents of XX* and if they are less than the contents of \( N_1(M) \) and \( N_1 + 1(M) \) branch to \( N_2 \).

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>X or XX*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>( N_1(M), N_2 )</td>
</tr>
<tr>
<td>Operand</td>
<td></td>
</tr>
</tbody>
</table>

If X is specified, \( N_1 \) may be:

1. A relative expression which refers to a lower data location, e.g.
   
   (a) A symbolic name referring to a lower data location.
   
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   
   (a) A decimal integer in the range 0 to 4095.
   
   (b) An octal integer in the range #0 to #7777.
   
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal.

\( N_2 \) may be:

1. A relative expression which refers to a program instruction, e.g.
   
   (a) A symbolic name that is the label of a program instruction.
   
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. A relative operand.

The operand refers to two locations. The first, \( N_1(M) \), is the location whose contents are compared with the contents of \( X \) and the second, \( N_2 \), is the program location to which control is transferred if the test conditions are satisfied.

If \( XX* \) is specified, \( N_1 \) may be types 1 and 2 given above for \( N_1 \) when \( X \) is specified; \( N_2 \) is as above. The operand refers to three locations. \( N_1(M) \) is the first of two consecutive locations whose contents are compared with the contents of \( XX* \), and \( N_2 \) is the program location to which control is transferred if the test conditions are satisfied.

In PLAN 4 a colon may be inserted between the comma and \( N_2 \) (see Note 5 below).

Execution

In the case of single-length locations, \( X \) is specified and the BXL macro-instruction will generate two machine-code instructions. Its effect is to compare the contents of \( X \) with the contents of \( N_1(M) \). Both words are considered as 24-bit positive numbers and thus no special significance is attributed to the sign bit. If C is initially clear and the contents of \( X \) are less than the contents of \( N_1(M) \), control is transferred to the point in the program specified by \( N_2 \). If C was left set by the previous instruction and the words being compared are equal, control is transferred to \( N_2 \). Otherwise the program continues with the next instruction in sequence. In the case of double-length locations, \( XX* \) is specified and the BXL macro-instruction will generate three machine-code instructions. Its effect is to compare the contents of \( XX* \) with the contents of \( N_1(M) \) and \( N_1 + 1(M) \). Both pairs of words are considered as 48-bit positive binary numbers and thus no special significance is attached to the sign bit.
If C is initially clear and the contents of XX* are less than the contents of \(N_1(M)\) and \(N_1 + I(M)\), control is transferred to the point in the program specified by \(N_2\).

If C was left set by the previous instruction and the words being compared are equal control is transferred to \(N_2\). Otherwise the program continues with the next instruction in sequence.

\(C\) is used and will be left clear (see Note 1).

\(V\) is not used and remains unchanged.

**Modification** This statement has an M-field. When modified, the least significant 15 bits of \(N + M\) are taken as the operand. In the extended data mode, the least significant 22 bits of \(N + M\) are taken as the operand.

**Examples**

1. It is required to test whether the contents of RULER are less than the contents of WIND. If they are, branch to DEPOS.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDX 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RULER</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SXL 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WIND,DEPOS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The macro-instruction will cause the following series of instructions to be generated in machine code:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TXL 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WIND</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEPOS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. It is required to test whether a double-length number in COVER and COVER + 1 is less than a double-length number in BOUT and BOUT + 1. If it is, branch to HEAD.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDX 56</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COVER</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SXL 56</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BOUT,HEA D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The macro-instruction will cause the following series of instructions to be generated in machine code:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TXL 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BOUT,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TXL 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BOUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEPOS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Machine Code Listing**

When \(X\) is specified: 027 X M \(N_1\)

074 -5 \(N_2\)

When XX* is specified: 027 X* M \(N_1 + 1\)

027 X M \(N_1\)

074 5 \(N_2\)

\(N_2\) is the label to which control is transferred.)
Notes

1 The Carry register (C) should be clear before the BXL macro-instruction is obeyed, except in testing quantities of more than double-length, where one or more TXL's must be used, and C may thus be set before reaching the BXL (see under TXL).

2 The fact that the words compared are considered as 24- or 48-bit positive binary integers makes it possible to use the BXL statement with alphanumerics data. No problem arises when using it to compare numbers with the same sign. If the numbers being compared have opposite signs, however, it is necessary to invert the sign bits of both numbers before making the comparison.

3 \[ \text{BXL X N}_1 (M), N_2 \]
   is equivalent to
   \[ \text{TXL X N}_1 \text{ M} \]
   \[ \text{BCS N}_2 \]
   \[ \text{BXL XX* N}_1 (M), N_2 \]
   is equivalent to
   \[ \text{TXL X* N}_1 +1 \text{ M} \]
   \[ \text{TXL X N}_1 \text{ M} \]
   \[ \text{BCS N}_2 \]

4 If X is specified and operand \( N_1 \), type I(a), is an undefined symbol, the compiler will allocate a word in lower data to the symbol.

   If XX* is specified, and operand \( N_1 \), type I(a), is an undefined symbol, the compiler will allocate two consecutive locations in lower data (the symbol is given the value of the first location).

5 In PLAN 4, the BCS instruction generated by this macro-instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the \( N_2 \) operand denotes that the BCS instruction generated by this macro-instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
Branch if X unequal to N(M)

**Function**

Compare the contents of X with the contents of N\(_1\) (M) and if unequal branch to N\(_2\).

Compare the contents of XX* with the contents of N\(_1\) (M) and N\(_1\) + 1(M); if unequal branch to N\(_2\).

**Format**

- **Operation Code**: BXU
- **Accumulator**: X or XX*
- **Operand**: N\(_1\) (M), N\(_2\)

If X is specified, N\(_1\) may be:

1. A relative expression which refers to a lower data location, e.g.
   - A symbolic name referring to a lower data location.
   - A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - A decimal integer in the range 0 to 4095.
   - An octal integer in the range 0 to 7777.
   - A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal.

N\(_2\) may be:

1. A relative expression which refers to a program instruction, e.g.
   - A symbolic name that is the label of a program instruction.
   - A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. A relative operand.

The operand refers to two locations. The first, N\(_1\) (M), is the location whose contents are compared with the contents of X, and the second, N\(_2\), is the program location to which control is transferred if the test conditions are satisfied.

If XX* is specified, N\(_1\) may be types 1 and 2 given above for N\(_1\); when X is specified, N\(_2\) is as above. The operand refers to three locations. N\(_1\) (M) is the first of two consecutive locations whose contents are compared with the contents of XX*, and N\(_2\) is the program location to which control is transferred if the test conditions are satisfied.

In PLAN 4, a colon may be inserted between the comma and N\(_2\) (see Note 4 below).

**Execution**

In the case of single-length locations, X is specified and the BXU macro-instruction will generate two machine-code instructions. Its effect is to compare the contents of X with the contents of N\(_1\) (M). If they are unequal, or if C was left set by the previous instruction, control is transferred to the point in the program specified by N\(_2\). Otherwise the program continues with the next instruction in sequence.

In the case of double-length locations, XX* is specified and the BXU macro-instruction will generate three machine-code instructions. Its effect is to compare the contents of XX* with the contents of N\(_1\) (M) and N\(_1\) + 1(M). If they are unequal, or if C was left set by the previous instruction, control is transferred to the point in the program specified by N\(_2\). Otherwise the program continues with the next instruction in sequence.
C is used and will be left clear (see Note 1).

Y is not used and remains unchanged.

**Modification** This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Examples**

1. It is required to compare the contents of LIGHT and HANG and if they are unequal to branch to LEAD.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>ACC</th>
<th>C</th>
<th>X</th>
<th>M</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>N'</th>
<th>H</th>
<th>D</th>
<th>Y</th>
<th>B</th>
<th>C</th>
<th>A</th>
<th>OPRAND</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX  5</td>
<td>LIGHT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BXU  5</td>
<td>HANG, PLEAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   The macro-instruction will cause the following instructions to be generated in machine code:

   | Label | Operation | ACC | C | X | M | N | Z | V | N' | H | D | Y | B | C | A | OPRAND | M |
   |-------|-----------|----|---|---|---|---|---|---|---|---|---|---|---|---|---------|----|
   | TXU  5 | HANG      |    |   |   |   |   |   |   |   |   |   |   |   |   |         |   |
   | BCS   | LEAD      |   |   |   |   |   |   |   |   |   |   |   |   |   |         |   |

2. It is required to compare the contents of a double-length number in MAR and MAR + 1 with a double-length number in SPOIL and SPOIL + 1. If they are unequal, branch to CATAS.

   | Label | Operation | ACC | C | X | M | N | Z | V | N' | H | D | Y | B | C | A | OPRAND | M |
   |-------|-----------|----|---|---|---|---|---|---|---|---|---|---|---|---|---------|----|
   | LDX  6 | MAR       |    |   |   |   |   |   |   |   |   |   |   |   |   |         |   |
   | BXU  6 | SPOIL, VCATAS |   |   |   |   |   |   |   |   |   |   |   |   |   |         |   |

   The macro-instruction will cause the following instructions to be generated in machine code:

   | Label | Operation | ACC | C | X | M | N | Z | V | N' | H | D | Y | B | C | A | OPRAND | M |
   |-------|-----------|----|---|---|---|---|---|---|---|---|---|---|---|---|---------|----|
   | TXU  7 | SPOIL, +1 |    |   |   |   |   |   |   |   |   |   |   |   |   |         |   |
   | TXU  6 | SPOIL     |    |   |   |   |   |   |   |   |   |   |   |   |   |         |   |
   | BCS   | CATAS     |   |   |   |   |   |   |   |   |   |   |   |   |   |         |   |

**Machine Code Listing**

When X is specified: 026 X M N₁
074 5 N₂

When XX* is specified: 026 X* M N₁ + 1
026 X M N₁
074 5 N₂

(N₂ is the label to which control is transferred.)

**Notes**

1. The Carry register (C) should be clear before the BXU macro-instruction is obeyed, except in testing quantities of more than double-length, where one or more TXU's must be used and C may thus be set before reaching the BXU (see under TXU).
2 BXU X N₁(M), N₂
   is equivalent to
   TXU X N₁(M)
   BCS N₂

BXU XX* N₁(M), N₂
   is equivalent to
   TXU X* N₁, 1(M)
   TXU X N₁(M)
   BCS N₂

3 If X is specified and operand N₁, type 1(a), is an undefined symbol, the compiler will allocate a
   word in lower data to the symbol.

If XX* is specified and operand N₁, type 1(a), is an undefined symbol, the compiler will allocate
   two consecutive locations in lower data (the symbol is given the value of the first location).

4 In PLAN 4, the BCS instruction generated by this macro-instruction is compiled as a direct branch,
   a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8.
   A colon inserted as the first character of the N₂ operand denotes that the BCS instruction generated
   by this macro-instruction is to be compiled as a replaced branch where it would otherwise be
   compiled as a relative branch; if the instruction is compiled to operate in direct branch mode,
   the colon is ignored.
Function  Test the contents of X and if they are zero transfer control to the location specified in N.

Format  

| Operation Code | BZE  
| Accumulator | X  
| Operand | N  

N may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand N refers to the program location to which control is transferred if X is equal to zero.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 1 below).

Execution
The BZE instruction tests the content of X and if it is zero causes a branch to the location specified by N. If the content of X is not zero, the program continues to the next instruction. The contents of X will remain unchanged.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification  The BZE statement has no M-field.

Machine Code Listing
050  X N

Notes
1. In PLAN 4, the BZE instruction is compiled as a direct branch, a relative branch or a replaced branch as determined by the rules stated on page 32 of Chapter 8. A colon inserted as the first character of the operand field denotes that the BZE instruction is to be compiled as a replaced branch where it would otherwise be compiled as a relative branch; if the instruction is compiled to operate in direct branch mode, the colon is ignored.
CALL

Subroutine Entry

**Function**
Perform the subroutine named in the operand before proceeding with the next instruction.

**Format**
- **Operation Code**: CALL
- **Accumulator**: X
- **Operand**: N

N may be:
1. A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. A relative operand.

The operand N refers to the subroutine to be performed.

In PLAN 4, N may be preceded by a colon as the first character of the operand field (see Note 9 below).

**Execution**
The CALL instruction causes control to be transferred to the location specified in the operand field. The address of the location immediately following the CALL instruction (the link address), and the state of V and zero suppression mode, are automatically stored in the accumulator X (the link accumulator). The format of the link accumulator depends on the address mode and branch mode in which the program member is operating (see Note 5). The contents of the link accumulator should be preserved while the subroutine called is being performed, in order to provide for re-entry to the main program (see the EXIT instruction).

**C** is not used and will be left clear.

**V** is used (see above) and will be left clear.

**Modification**
This statement has no M-field.

**Example**
The first section of program shown below calls the subroutine CONV. The second section of program shows the subroutine and the EXIT back to the main program.

| Label | Operation | Acc | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | **Address** |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|72|128|
| CALL | CONV | 4, K, Y, Z | PARAMETERS | 7, 7, 6 | PARAMETERS | 8, 8 | PARAMETERS | 0, REST | INSTRUCTION |

| Label | Operation | Acc | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | **Address** |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|72|128|
| CONV | LD X | 0 | 0 (1) | PARAMETERS | F, K, U, P, FIRST | PARAMETER |

Other Instructions.

| Label | Operation | Acc | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | **Address** |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|72|128|
| EXIT | 3 |

The '3' written in the operand field of the EXIT instruction will cause the main program to be re-entered, not at the address specified in the link accumulator XI but at an address three locations greater.
Machine Code Listing
070 X N

Notes

1 The return to the main program is effected by means of the EXIT instruction (q.v.) specifying the
link accumulator, as the last instruction of the subroutine.

2 A subroutine may require parameters to be set before the routine is performed. These parameters
 can be held in the locations immediately following the CALL instruction. The first parameter will
be held in the location addressed by the link, and may be accessed from the subroutine by using
the link accumulator as a modifier. (See the Example on page 6.)

3 Library subroutines are accessed by specifying the name of the subroutine in the operand field of
the CALL instruction. These subroutines will require parameters (if any) to be specified as
 described in Note 2. The CALL instruction must utilize the accumulator specified in the subroutine
 library specification. The library subroutine will be incorporated into the program during the
consolidation pass.

4 If the CALL instruction is used to transfer control to a subroutine outside the given segment, the
operand must be a symbolic name not used within the segment. It may be either a subroutine
segment name (if control is to be transferred to the first program instruction in the subroutine) or
a symbolic name which has been defined in the subroutine segment with a #CUE directive (if
control is to be transferred to the instruction labelled by this #CUE).

5 Information is stored in the link accumulator as follows:
   (a) if the program member is operating in compact mode and direct branch mode (15AM and DBM):
   
<table>
<thead>
<tr>
<th>Bit 0</th>
<th>The state of V when the CALL instruction was obeyed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 1 to 7</td>
<td>Undefined.</td>
</tr>
<tr>
<td>Bit 8</td>
<td>The state of zero suppression mode when the CALL instruction was obeyed.</td>
</tr>
<tr>
<td>Bits 9 to 23</td>
<td>The link address.</td>
</tr>
</tbody>
</table>

   No assumptions should be made about bits 1 to 7, nor should they be altered in any way.

   (b) if the program member is operating in extended data mode (22AM), or in compact mode and
   extended branch mode (15AM and EBM):
   
<table>
<thead>
<tr>
<th>Bit 0</th>
<th>The state of V when the CALL instruction was obeyed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 1</td>
<td>The state of zero suppression mode when the CALL instruction was obeyed.</td>
</tr>
<tr>
<td>Bits 2 to 23</td>
<td>The link address.</td>
</tr>
</tbody>
</table>

6 If, when the CALL instruction came to be obeyed, zero suppression mode was set to 0, it is left
set to 0. If, when the CALL instruction came to be obeyed, zero suppression mode was set to 1,
it is left in an indeterminate state.

7 On returning by means of an EXIT instruction, the state of zero suppression mode is restored, and
the return address is determined, in accordance with the information then standing in the link
accumulator which is interpreted in accordance with the operating modes applying at that time. It is
therefore inadvisable to enter a subroutine in one address mode and leave it in another, as, in
those circumstances, not only may the zero suppression mode be incorrectly restored, but also
the link address may be wrongly interpreted. Similarly, if the program member is operating through-
out in compact mode, it is inadvisable to enter a subroutine in one branch mode and leave it in
another.

8 If the CALL instruction is the operand instruction of an OBEY instruction, then the link address
stored in the accumulator X is that of the location following the OBEY, not that of the location
following the CALL. If there is a chain of OBEY instructions leading to the CALL, the link address
stored is that of the location following the first OBEY of the chain.

9 In PLAN 4, the CALL instruction is compiled as a direct branch, a relative branch or a replaced
branch as determined by the rules stated on page 32 of Chapter 3. A colon inserted as the first
character of the operand field denotes that the CALL instruction is to be compiled as a replaced
branch where it would otherwise be compiled as a relative branch; if the instruction is compiled
to operate in direct branch mode, the colon is ignored.

Chapter 4  65
Convert Binary to Decimal

Function

Convert a binary fraction in X and X* to a decimal character in N(M).

Format

- **Operation Code**: CBD
- **Accumulator**: X
- **Operand**: N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) refers to the location in which the decimal character will be stored.

Execution

The CBD instruction multiplies the positive double-length binary fraction in X and X* by ten. The integral part of the product is then stored as a decimal character in the position specified by N(M), and the fractional part is left in XX*.

An incorrect result will be given if B0 of X or X* is initially 1. Non-significant zeros in the resultant decimal number will be replaced by spaces if MODE has been set to 1. If the integral part of the product is not zero, MODE is set equal to 0 so that further zeros are treated as significant (see MODE).

C is not used and will be left clear.

V is not used and remains unchanged.

**Modification**

This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand. See Note 1 below.

Examples

1. It is required to convert a double-length binary fraction in X6 and X7 into a decimal fraction with twelve places of decimals (including leading zeros). The result is to be stored in RESL + 2.2 onwards.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>zee</th>
<th>zee</th>
<th>zee</th>
<th>zee</th>
<th>zee</th>
<th>OFFERD</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX</td>
<td>2</td>
<td>12</td>
<td>RESL</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td>L0ADV8</td>
</tr>
<tr>
<td>CBD</td>
<td>6</td>
<td>0(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCHX</td>
<td>2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
It is required to convert a non-negative binary integer in INT into four decimal characters. The character string is to be loaded into RESL. ITEM holds 10^4.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>ACC</th>
<th>OP 1</th>
<th>OP 2</th>
<th>OP 3</th>
<th>OP 4</th>
<th>Label</th>
<th>OP 1</th>
<th>OP 2</th>
<th>OP 3</th>
<th>OP 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX</td>
<td>5</td>
<td>INT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LOAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APN</td>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CLEAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVR</td>
<td>5</td>
<td>ITEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DIVIDE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDCT</td>
<td>7</td>
<td>#200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ROUNDU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDS</td>
<td>1</td>
<td>1</td>
<td>RESL</td>
<td></td>
<td></td>
<td></td>
<td>LOAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBD</td>
<td>6</td>
<td>0(1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CONVER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCHK</td>
<td>1</td>
<td>#1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BRANCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Machine Code Listing

047 X \ M N

Notes

1. If a CBD statement is unmodified, the decimal integer generated is put in the n3 position. This is true whether or not there is supplementary modification by a preceding SM0 instruction.
2. If a binary integer is to be converted it must first be divided by 10^n, where n is the number of characters required in the final form. The result of the division must be rounded up (even if a DVR function is used).
3. A double-length binary fraction must not exceed a maximum of 14 decimal digits, or a single-length binary fraction 7 decimal digits.
4. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
5. A CBD instruction is undefined if X = N(M) or X^* = N(M).
CDB

Convert Decimal to Binary

**Function**  Convert a decimal character in N(M) to binary form in X and X*.

**Format**

- **Operation Code**: CDB
- **Accumulator**: X
- **Operand**: N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range 0 to 7777.
   - (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to the location which contains the character to be converted.

**Execution**

The CDB instruction multiplies the double-length binary number in X and X* by ten and adds in the character specified in N(M), having first checked that this character is numeric.

If the specified character is not in the range 0 to 9, XX* and V will be unchanged, and C will be set.

If B0 of X is originally 1, V may be set and an incorrect answer given in X; the answer in X* is, however, unaffected by the previous contents of X. If B0 of X* is originally 1, an incorrect answer will be given.

- **C**: not used but will be set if a non-numeric character is encountered, otherwise left clear.
- **V**: will be set if XX* contains more than the representation of 1/10 as a double-length fraction, otherwise left unchanged.

**Modification** This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 32 bits of N + M are taken as the operand. See Note 1 below.

**Examples**

1. It is required to convert to binary form a string of nine numeric characters starting at DEC + 1.2

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDN</td>
<td>6</td>
<td>Q</td>
<td>I_CLEARVX</td>
</tr>
<tr>
<td>LDN</td>
<td>7</td>
<td>Q</td>
<td>I_CLEARVX*</td>
</tr>
<tr>
<td>LDX</td>
<td>2</td>
<td>'9/DEC+1.2'</td>
<td>L_LOADYCONTROLWORD</td>
</tr>
<tr>
<td>CDB</td>
<td>6</td>
<td>0(2)</td>
<td>I_CONVERT1NEXTCHARACTER</td>
</tr>
<tr>
<td>BCCHK</td>
<td>2</td>
<td>0=1</td>
<td>I_BRANCHY1NEXTCHARACTER</td>
</tr>
</tbody>
</table>

68
It is required to convert to binary form a string of numeric characters. The string is of unknown length but is terminated by a non-numeric character. The first character is in QUANT + 2.1.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>0</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDM</td>
<td>6.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX</td>
<td>7.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDB</td>
<td>6</td>
<td>0(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCHX</td>
<td>2</td>
<td>n+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It should be noted that though the counter-modifiers in the above examples are loaded by using literals, this facility is not available in PLAN 1. Instead it would be necessary to define these control words as items of constant data under a #LOWER directive.

**Machine Code Listing**

043 X M N

**Notes**

1. If the instruction is unmodified, the integer in the n3 position is converted. This is true whether or not there is supplementary modification by a preceding SMO instruction.

2. All decimal digits converted are regarded as digits of an integer. If the number is in fact mixed, or is a fraction, then, after the conversion loop has been left, the result must be divided by 10^n where n is the number of decimal digits after the point.

3. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

4. A CDB instruction is undefined if X = N(M) or X* = N(M).
CONT

Read in More Program

Function

Suspend the program, and read in more program from the specified peripheral unit.

Format

Operation Code

CONT

The rest of the instruction can take three possible forms

1 Accumulator

Blank

Operand

The symbolic name of the relevant peripheral (CR0, TR2, etc.)

2 Accumulator

The program's unit number of the relevant peripheral.

Operand

N(M), where N(M) is the type number of the peripheral.

0 = Paper Tape Reader

3 = Card Reader

5 = Magnetic Tape

12 = Cassette Tape.

3 Accumulator

X, where X is an accumulator whose least significant six bits contain

the program's unit number of the relevant peripheral.

Operand

N(M), where N(M) = 256 + the type number of the peripheral as specified in 2 above.

PLAN 1 Compilers will not accept operands of type 1.

Execution

The CONT instruction causes the program to be suspended immediately. Executive then reads binary program (which must be in standard format, but must not include a request block) from the specified peripheral into the program's area, the locations overwritten being determined by the data blocks read. The transfer terminates when an entry block is read.

The action taken on completion of the loading will depend on the type of entry block read. Usually this will be of type 4, which will cause the program to continue with the instruction in the location following that which contained the CONT instruction (either or both of these locations may have been overwritten by the program just read in).

C is not used and will be left clear.

V is not used and remains unaffected.

Modification

This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand. See Note 1 below.

Examples

| LABEL | OPERATION | ACC | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| CONT  |           |     |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| CONT   | E.R1      |     |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| CONT   | 3         |     |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| CONT   | E.59      |     |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| CONT   | E.256(2)  |     |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

In each case further program will be read in from the card reader CR1.

Machine Code Listing

154 X M N

70
Notes

1 The operand is modifiable, except in the case of a type 1 operand. If the operand is of types 2 or 3, the unit type may be modified. Note that a type 2 operand would be interpreted as a type 3 operand if B15 was set in the modifier.

2 If a type 3 operand is used, bits 0 to 17 of the accumulator specified by X must be set to zero.

3 If the program consists of more than one member, the CONT instruction is valid only if initiated by member 0. It causes all members of the program to be suspended immediately.

4 The instruction is illegal if:
   (a) the specified device is not of a suitable type (suitable types include Card Readers, Paper Tape Readers, and certain File Storage devices),
   (b) the specified unit is not assigned to the program,
   (c) it is issued by a member of the program other than member 0.

5 Errors detected during the progress of a CONT instruction (other than the illegal conditions mentioned above) will produce the same effect as if the loading had been initiated by the console directive 'LOAD' (see the Console Operating Manual appropriate to the processor concerned). It must be noted, however, that should a loading error occur, typing 'GO' will not cause the CONT instruction to be repeated, hence a major restart must be carried out in such cases.

6 Each data block read itself specifies, in the second word of the block, the destination address, relative to datum, for the first word of program in the block. The areas overwritten may be anywhere in the program area, except for the reserved words. Program loaded by the CONT instruction cannot extend beyond the program's limit.

7 The area read into is not zeroized before loading, hence care must be taken in respect of areas which were zero when the binary program was created. If the binary program was created by a dump of type 0 data blocks, the dump will have omitted any blocks in which all 16 program words were zero.

8 When a CONT instruction is initiated, Executive starts the loading process without checking the state of any peripheral devices, other than the one to be used in the loading process. If, therefore, any other peripheral units are assigned to the program and are active at the time the instruction is given, the result is undefined. This restriction does not apply to magnetic tape rewinds or cassette tape alignments, provided that the reply words for these are not within the area being overlaid.

9 If any flag setting peripheral is assigned to the program, the result of a CONT instruction is undefined.

10 Information on binary program formats, and on the various kinds of request, data and entry blocks, may be found in Chapter 7 of the Central Processors manual.
CLOSE

Close a Magnetic Tape

Function
Closes a magnetic tape.

Format
Operation Code: CLOSE
Accumulator: A decimal digit in the range 0 to 7, specifying the program's unit number for the magnetic tape unit.
Operand: Blank

Execution
CLOSE causes the tape on a specified unit to be closed, rewound and released from the program.

Modification
The CLOSE instruction has no M-field.

Machine Code Listing
157 X 0 n LT
where n is the relative address in the literal table of the first of two consecutive locations which hold a control area of form:

  first word: 5/ #1000
  second word: 0

Notes
1 This is one of a set of magnetic tape macro-instructions which permit the execution of simple basic functions on magnetic tape without requiring the programmer to set up control areas. (See also BSP, BTM, FTM, REW, SCR, UNL, WTM.)
**DCH**

**Deposit Character**

**Function**
Deposit a character from X into the location specified by N(M).

**Format**
- **Operation Code**: DCH
- **Accumulator**: X
- **Operand**: N(M)

If X is specified, N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the word into which the character is to be deposited.

**Execution**

In its unmodified form the DCH statement will make bits 18 to 23 (the least significant character) of N equal to bits 18 to 23 of X. The remaining bits of N and all of X are left unchanged.

The DCH instruction may be modified by an accumulator whose contents take the form of a character index word (see BCHX).

When so modified the instruction deposits the character from bits 18 to 23 of X into the word specified by the sum of N and bits 9 to 23 of the index word, at the character position (n0 to n3) specified by bits 0 to 1 of the index word.

C is not used and will be left clear.

V is not used and remains unchanged.

**Modification**
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Examples**

1. X1 holds a character index word: 0/4.2

X6 holds:

| 011010 | 110010 | 110111 | 101101 |

FIN + 4 holds:

| 011101 | 000001 | 010101 | 110110 |

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>X10</th>
<th>X09</th>
<th>X08</th>
<th>X07</th>
<th>X06</th>
<th>X05</th>
<th>X04</th>
<th>X03</th>
<th>X02</th>
<th>X01</th>
<th>X00</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BCM</td>
<td>6</td>
<td>FIN(4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation: 

Result in FIN + 4:

| 011101 | 000001 | 101101 | 110110 |

This example has caused character n3 of X to be deposited into the n2 position of N(M).
Eight characters stored successively from the n2 position of DATA + 3 to the n1 position of DATA + 5 are to be stored in an area beginning at the n1 position of ITEM, both areas being in lower data.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>Y.D.</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDIX</td>
<td>1</td>
<td>'8/3'</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX</td>
<td>2</td>
<td>'8/01'</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDCH</td>
<td>4</td>
<td>DATA(1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCHX</td>
<td>1</td>
<td>ITEM(2)</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCHX</td>
<td>2</td>
<td>+3</td>
<td>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This example illustrates the use of both the LDCH and DCH instructions. The first instruction loads a character index word into X1 to control the extraction of eight characters from store. The second instruction loads another character index word into X2 to control the insertion of eight characters into store. The LDCH instruction extracts a character from DATA (1) and inserts it in X4 (e.g. the first time this instruction is obeyed the n2 character of DATA + 3 will be loaded into the n3 position of X4).

The DCH instruction inserts the n3 character of X4 into ITEM (1) (e.g. the first time the instruction is obeyed the n3 character of X4 is loaded into the n1 position of ITEM).

The two BCHX instructions ensure that the index words are advanced, so that successive characters are extracted and inserted each time the instructions are obeyed.

**Machine Code Listing**

034 X M N

**Notes**

1 Though a character may be loaded into any of the four character positions of N(M) by the modified form of a DCH statement, the character transferred from X is always n3. Note too, that a DCH statement with a zero modifier has a different effect from an unmodified DCH. This is true whether or not there is a supplementary modifier; supplementary modification by a SMO instruction does not affect the character position specified.

2 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
Delete the Object Program

Function
Delete the program and type the message DELETED, followed by a two-character code.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>DEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>This may be:</td>
</tr>
<tr>
<td></td>
<td>1 Up to four octal digits (modifiable).</td>
</tr>
<tr>
<td></td>
<td>2 Two symbols which can be represented in alpha shift preceded by 2H (unmodifiable); not PLAN 1.</td>
</tr>
</tbody>
</table>

Execution
The DEL instruction causes a voluntary entry to Executive which then types a message on the console typewriter. The message includes the word DELETED (or an abbreviation thereof) followed by two characters from the card/printer set. The characters $, ], {, and − (#74 to #77) may not be used.

These two characters are formed as follows:

1 If the operand was written as up to four octal digits, preceded by # and possibly modified, then the corresponding 12-bit quantity (if modified, only the least significant 12 bits are used) is treated as two 6-bit characters and the corresponding symbols are printed.

2 If the operand was written as two symbols preceded by 2H, then these symbols are printed (not PLAN 1).

On dual and multiprogramming processors, the message is preceded by n#NAME, where n is the program member number and NAME is the name of the program sending the message. The program is then deleted from the store. In a multiprogram (but not single or dualprogram) processor, any programs occupying higher addresses in the store are moved to fill the store area vacated by the program just deleted.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand. Bits above bit 12, which can only arise through modification, will be ignored in the output message. (See Execution.)

Examples

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>OPERAND</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEL</td>
<td>#3241</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEL</td>
<td>#3202(1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEL</td>
<td>2HAB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first two would both result in this message being typed (assuming the program name to be #INVO):

0#INVO; DELETED: − *A

The third would result in the message:

0#INVO; DELETED: − AB

Machine Code Listing
161 2 M N

4222(7.72)
Notes

1. The compiler inserts 2 in the accumulator field of the machine-code instruction generated to distinguish it from SUSWT and DISP.

2. The operand N is listed as the decimal equivalent of the two characters or octal number written in the operand field.

3. On a 1901 central processor without a console typewriter, the 12-bit equivalent of the two characters is displayed on the control panel lights.

4. The effect of a DEL instruction upon the position of the data bearing medium of any devices still assigned to the program is undefined; for example, a magnetic tape may or may not be rewound.
DELTY

PLAN 1, 2, 3, 4

Delete and Type

Function

Delete the program and type a variable-length message to be treated by Executive as a console typewriter input message.

Format

Operation Code
DELTY

Accumulator
Blank

Operand
N(M)

N may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

3 A literal (not PLAN 1).

The operand refers to a location which holds an index word consisting of a count of the number of characters (maximum 40) and the address of the first character of the message (which must be n0).

Execution

The DELTY instruction deletes the program and types a message on the console typewriter. The message must be either a FIND directive or a LOAD directive. Executive then operates upon the message as if it had been input by the operator, except that the information in the program's accumulators at the time of deletion is passed on to the accumulators of the successor program. (See further notes 6 and 10.) The DELTY instruction should only be used in member 0 of the program.

The message may be up to 40 characters long, and is referenced by means of an index word. It must not contain the characters $, ], ^ or ( #74 to #77 ). The index word, held in the location specified by N(M), indicates the number of characters in the message and the address of the first character. The first character must be in the n0 position of a word.

On dual and multiprogramming processors, when the message is output on the console typewriter, it is preceded by 0#name; DELETED: - (or an abbreviation thereof) where name is the name of the program sending the message. On single-program processors, the message is preceded by DELETED only (or an abbreviation thereof).

C is not used and will be left clear.

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N+M are taken as the operand. In the extended data mode, the least significant 22 bits of N+M are taken as the operand.

Execution in Extended Data Mode

When operating in extended data mode the operand N(M) may refer to either of the following:

1 A location which holds an index word as defined above.

2 A location which is the first word of a two word control area. The format of the control area is:
Word N(M) Bits 0 to 17 zero
Bits 18 to 23 character count
Word N+1(M) Bits 0 to 1 zero
Bits 2 to 23 address of the word containing the first character.

Whether the instruction is to use a one word or a two word control area is determined by whether or not bits 0 to 9 of word N(M) are zero.

Example
When a program has finished it is required to delete it, to search unallocated magnetic tape units for a tape whose header label contains the file name 'PROGRAM LOLA' and to load this program into store. This could be achieved by the following sequence:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>12 1</th>
<th>20 24</th>
<th>32</th>
<th>40 44</th>
<th>52</th>
<th>60 64</th>
<th>72</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>#LOWER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESSG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#PROGRAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELTY</td>
<td>COUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERR1</td>
<td>MESSG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Machine Code Listing
160 2 M N

Notes
1 The compiler inserts 2 in the accumulator field of the machine-code instruction generated to distinguish it from the SUSTY and DISTY instructions.

2 The use of this instruction on a 1901 central processor without a console typewriter is not recommended since if it is used, only the index word will be displayed on the control panel lights. The program will be deleted, but the instruction will not result in the loading of the successor program.

3 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

4 Unless the program is being run under the control of a GEORGE operating system, the program name contained in the FIND or LOAD message must not be that of a program already in the machine.

5 If the message typed is acceptable to Executive, the normal Executive response OK is appended. If the message is not acceptable then the appropriate error message will be typed.

6 The contents of the accumulators of the deleted program will be carried forward to the accumulators of the successor program only if the FIND or LOAD message is implemented successfully. If the message is not implemented successfully and an operator FIND or LOAD has to be used, then the contents of the accumulators will be lost.

7 If the LOAD form of the message is used, then, as with console typewriter LOAD messages, the absolute unit number of the load peripheral must be given. The F button (or Control and A) "Load on unit 0" facility, whereby Executive will imply the device number of the Executive load device, is not available with DELTY instructions. If the unit number specified is that of a peripheral which is attached to the program, and the message has satisfied Executive's validity checks, then, although released from the program being deleted, the unit will not be made available for assignment to another program but will effectively be assigned to Executive until the loading is completed. The data on the specified peripheral will not be repositioned: in particular, if the unit is a magnetic tape deck the tape will not be rewound, therefore the program being deleted should leave the tape positioned so that the next block read from it will be the successor program's request block. (This does not apply if the FIND form of message is used.)
A program should not delete itself if its peripherals are busy. If at the time a DELTY instruction is given the reply word to a PERI instruction indicates an operation in progress, then that operation may never be successfully completed. If the PERI instruction was applied to the device specified as the LOAD device in the DELTY message, then the loading process may not take place successfully.

Files not closed at the time of deletion (other than a file on the load device, as indicated in Note 7) may be left in an indeterminate state; for example, magnetic tapes may or may not be rewound.

If the DELTY instruction is used for chaining programs, that is for passing information from one program to a logically related successor program via their accumulators, then care should be taken that the loading process does not overwrite the accumulators.

The successor program should always be loaded in binary form. If an attempt is made to chain a program held in G.P.L. load form, a loader program will intervene between the deleted program and its successor, and the attempt may fail.

The destination addresses of the binary program blocks of the successor program must not include locations 0 to 7. Note that binary dumps produced by Executive are not suitable for chaining because when loaded, the first block of the program would overwrite locations 0 to 7. With programs on paper tape or cards this may be overcome by removing the second block of paper tape or the second card from the binary dump, and adjusting the entry block accordingly. (For a description of the entry block and binary program formats see the ICL 1900 Series manual 'Central Processors', Chapter 7.) For programs being loaded from magnetic tape or cassette tape the requirement may be satisfied by using binary programs previously obtained by means of the appropriate overlay loader, #XPO6 or #XPP6 (see page 58 of Chapter 7).

For non-overlay programs on magnetic tape or cassette tape the overlay loader may be obtained, with those compilers in which the facility is implemented (see page 37 of Chapter 7), by the use of the BINARY steering line item. Binary programs on disc output by the consolidators #XPCK or #XPCL are in a form suitable for chaining.

Where the chained program is to be found from a magnetic medium, a search program may intervene. The standard ICL search programs #TAPE, #XPKP, #XPKQ, #XPEA, #XPED and #XPFB all preserve the contents of the accumulators and pass them on to the successor program.

Binary programs output by the consolidator #XPCL on to E.D.S. are in a form suitable for chaining.

If successor programs are to be loaded by a DELTY instruction from library tapes created by #XPMU or #XPMV or from library files created by #XPEU, the user should ensure when creating these files that the binary dumps of programs likely to be chained are adjusted suitably.
**DEX**

**Deposit Exponent**

**Function**
Store in \( \text{N}(\text{M}) \) the contents of bits 15 to 23 of X.

**Format**

- **Operation Code**: DEX
- **Accumulator**: X
- **Operand**: \( \text{N}(\text{M}) \)

\( \text{N} \) may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range \#0 to \#7777.
   - (c) A previously defined absolute symbol in the range 0 to 4095.

The operand \( \text{N}(\text{M}) \) refers to the store location in which the nine bits of X are to be stored.

**Execution**

The DEX instruction causes bits 15 to 23 of \( \text{N}(\text{M}) \) to be made equal to bits 15 to 23 of X. Bits 0 to 14 of \( \text{N}(\text{M}) \) and all of X are left unchanged.

- \( \text{C} \) is not used and is left clear.
- \( \text{V} \) is not used and remains unchanged.

**Modification**

This statement has an M-field. When modified, the least significant 15 bits of \( \text{N} + \text{M} \) are taken as the operand. In the extended data mode, the least significant 22 bits of \( \text{N} + \text{M} \) are taken as the operand.

**Example**

Original contents of X4: 110 110 001 100 010 110 010 100

Original contents of GRASP: 010 110 101 110 111 011 011 111

**Operation:**

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>XOR</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DEX</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Result in GRASP:** 010 110 101 110 111 010 010 100

**Machine Code Listing**

035 X M N

**Notes**

1. If operand type 1 (a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
Disengage Peripheral

Function
Disengage the specified peripheral, but do not make it available to other programs.

Format

Operation Code  
DIS

The rest of the instruction can take three possible forms:

1. **Accumulator**  
   Blank
   **Operand**  
The symbolic name of the relevant peripheral (LP0, TP2, etc.).

2. **Accumulator**  
   The program’s unit number of the relevant peripheral.
   **Operand**  
   N(M), where N(M) is the type number of the peripheral.
   - 0 = Paper Tape Reader
   - 1 = Paper Tape Punch
   - 2 = Line Printer
   - 3 = Card Reader
   - 4 = Card Punch
   - 20 = Digital Incremental Plotter

3. **Accumulator**  
   X, where X is an accumulator whose least significant six bits contain the program’s unit number of the relevant peripheral.
   **Operand**  
   N(M), where N(M) = 256 + the type number of the peripheral as specified in 2 above.

Execution
The specified peripheral is disengaged, so that it cannot be used in a subsequent transfer until the operator has re-engaged it by pressing the ALLOCATE button. The peripheral remains allotted to the program.

Before disengaging the unit Executive checks that it is not busy. If the unit is busy Executive suspends the program, removing the suspension and proceeding with the instruction when the peripheral is not busy.

If an attempt is made to initiate a transfer using a peripheral after it has been disengaged and before it has been re-engaged, the program is suspended, and Executive types a message on the console typewriter to notify the operator of the situation.

C  is not used and will be left clear.

V  is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand. See Note 1 below.

Examples

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>32</th>
<th>16</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>45</th>
<th>50</th>
<th>55</th>
<th>60</th>
<th>65</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIS</td>
<td>1</td>
<td>P3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIS</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIS</td>
<td>7</td>
<td>255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIS</td>
<td>7</td>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In each case the line printer LP3 is temporarily disengaged.

Machine Code Listing

152 X M N
Notes

1 The operand is modifiable, except in the case of a type 1 operand. If the operand is of type 2 or 3, the unit type may be modified. Note that a type 2 operand would be interpreted as a type 3 operand if B15 was set in the modifier.

2 If a type 3 operand is used, bits 0 to 17 of the accumulator specified by X must be set to zero.

3 A DIS instruction is illegal if it is given in respect of magnetic tape devices, file storage devices or flag-setting devices.

4 A basic peripheral can also be disengaged by the operator pressing the appropriate button.

5 The instruction may be used at a point in a program where a peripheral needs operator attention, e.g. changing paper on the line printer.
Function  Type the message DISPLAY followed by a two-character code.

Format  

Operation Code  DISP

Accumulator  Blank

Operand  This may be:

1  Up to four octal digits (modifiable).
2  Two symbols which can be represented in alpha shift preceded by 2H (unmodifiable); not PLAN 1.

Execution  
The DISP instruction causes a message to be typed on the console typewriter. The message includes the word DISPLAY (or an abbreviation thereof) followed by two characters from the card/printer set. The characters $, ], ^, and _-(#T4 to #T7) may not be used. The two characters are formed as follows:

1  If the operand was written as up to four octal digits, preceded by # and possibly modified, then the corresponding 12-bit quantity (if modified, only the least significant 12 bits are used) is treated as two 6-bit characters and the corresponding symbols are printed.
2  If the operand was written as two symbols preceded by 2H, then these symbols are printed (not PLAN 1).

On dual and multiprogramming processors, each message is preceded by n=NAME, where n is the program member number and NAME is the name of the program sending the message.

C  is not used and will be left clear.

V  is not used and cannot be set.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand. Bits above bit 12, which can only arise through modification, will be ignored in the output message. (See Execution.)

Examples  

The first example above would result in the message DISPLAY:- %? being typed out (preceded by the program name on a dual or multiprogramming processor).

Machine Code Listing  

161 1 M N

where N is the decimal value of the operand.

Notes  

1  The compiler inserts 1 in the accumulator field of the machine-code instruction generated to distinguish it from SUSWT and DEL.

2  On a 1901 central processor without a console typewriter, the 12-bit equivalent of the two characters is displayed on the control panel lights and the program is suspended.
DISTY

Display Message

Function
Type a variable-length message of up to 40 characters without suspending the program.

Format
- Operation Code: DISTY
- Accumulator: Blank
- Operand: N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.
3. A literal (not PLAN 1).

The operand N(M) refers to a location which holds an index word consisting of a count of the number of characters (maximum 40) and the address of the first character of the message (which must be character 0).

Execution

The DISTY instruction causes a message to be typed on the console typewriter. This message may be up to 40 characters long, and is referenced by means of an index word. It must not contain the characters represented by #74 to #77. The index word, held in the location specified by N(M), indicates the number of characters, and the address of the first character in the message. The first character must be in the n0 position of a word. On dual and multiprogramming processors, each message is preceded by nname; DISPLAY:-- (or an abbreviation thereof) where n is the program member number and name is the name of the program sending the message. On single-program processors, the message is preceded by DISPLAY only (or an abbreviation thereof).

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Execution in Extended Data Mode

When operating in extended data mode the operand N(M) may refer to either of the following:
1. A location which holds an index word as defined above.
2. A location which is the first word of a two word control area. The format of the control area is:
   - Word N(M) Bits 0 to 17 zero
     Bits 18 to 23 character count
   - Word N+1(M) Bits 0 to 1 zero
     Bits 2 to 23 address of the word containing the first character.

Whether the instruction is to use a one word or a two word control area is determined by whether or not bits 0 to 9 of word N(M) are zero.
Example

At a certain point in a program it is required to type out the message INVOICE CALCULATIONS COMPLETE; this could be achieved by the following sequence:

| LABEL     | OPERATION | ACC  | 12 | 56 | 20 | 28 | 52 | 30 | 40 | 60 | 64 | 68 | PROC
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESSG.</td>
<td>Z90/INVOICE/CALCULATIONS/COMPLETE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNT</td>
<td>29/MESSG.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#PROGRAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISTY</td>
<td>CNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Machine Code Listing

160 1 M N

Notes

1. The compiler inserts 1 in the accumulator field of the machine-code instruction generated to distinguish it from the SUSTY and DELTY instructions.

2. The use of this instruction on a 1901 central processor without a console typewriter is not recommended since, if it is used, only the index word will be displayed on the control panel lights, and the program will be suspended.

3. If operand type 1 (a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
**DLA**

**Deposit Long Address**

Function: Store in N(M) the long address contained in X.

Format:

- **Operation Code**: DLA
- **Accumulator**: X
- **Operand**: N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the store location in which the 15 bits of X will be stored.

**Execution**

The DLA instruction causes bits 9 to 23 of N(M) to be made equal to bits 9 to 23 of X. Bits 0 to 8 of N(M) and all of X are left unchanged.

C is not used and will be left clear.

V is not used and remains unchanged.

**Modification**

This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Examples**

1. Original contents of X5: 101 010 111 111 010 111 000 111
   
   Original contents of CORK: 111 011 001 000 101 111 010 010
   
   Operation:
   
   ![Operation Diagram]
   
   Result in CORK: 111 011 001 111 010 111 000 111
   
   (X5 is unchanged.)

2. The word CONS is in lower data and contains an index word '8/HEAD'. It is required to alter this word so that the long address (i.e. the least significant 15 bits) refers to location DATA (which is in lower data).

   ![Operation Diagram]
Machine Code Listing

037  X M N

Notes

1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
DSA

Deposit Short Address

Function  Store in N(M) the short address contained in X.

Format    Operation Code
           DSA

Accumulator
           X

Operand    N(M)

N may be:

1  A relative expression which refers to a lower data location, e.g.
(a)  A symbolic name referring to a lower data location.
(b)  A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.

2  An absolute expression in the range 0 to 4095, e.g.
(a)  A decimal integer in the range 0 to 4095.
(b)  An octal integer in the range #0 to #7777.
(c)  A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the store location in which the short address
is to be stored.

Execution

The DSA instruction causes bits 12 to 23 of N(M) to be made equal to bits 12 to 23 of X. Bits 0 to 11
of N(M) and all of X are left unchanged.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are
taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the
operand.

Examples

1  Original contents of X5: 111 101 101 101 101 101 110 001

  Original contents of WELL: 011 001 110 001 001 100 011 110

  Operation:

  Label  Operation  ACC
  1 11 4 7 0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60

  DSA      5       WELL

  Result in WELL: 011 001 110 001 101 101 110 001

  (X5 is unchanged.)

2  The address held in the least significant 12 bits of SALES which is in lower data, is to be replaced
by the number 1248.

  Operation:

  Label  Operation  ACC
  1 11 4 7 0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60

  LDN      5       1248

  DSA      5       SALES
Machine Code Listing

036 X M N

Notes

1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
Unrounded Double-length Divide

Function  Divide the double-length number in XX* by the single-length number in N(M).

Format  

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>DVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to a store location which contains the divisor.

Execution

The double-length dividend in X and X* is divided by the single-length number in N(M). The single-length quotient is left in X* and the remainder in X. The double-length dividend and the single-length divisor may be integers, fractions or mixed numbers. (If the numbers have fractional parts, the number of bits after the binary point in the quotient is equal to the difference between the number of bits after the binary points in the dividend and divisor.) Negative numbers are catered for and a correctly signed quotient is produced. A remainder is left in X which satisfies the following equation: Remainder = Dividend - Quotient x Divisor. The remainder will always have the same sign as the divisor. The contents of N(M) are unaltered.

C is not used and will be left clear.

V will be set for the following conditions:

1. If the quotient exceeds the capacity of one computer word.
2. If an attempt is made to divide by zero, in which case the dividend is unchanged in XX*.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

A length in inches is stored double-length in INS and INS + 1. It is required to convert this quantity to yards, feet and inches and store in locations YARDS, FEET and INS. The conversion factors 12 and 3 are held in CON12 and CON3. Assume that the quotient of the first division can be stored in one word.
Machine Code Listing

044 X MN

Notes

1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

2. Considering the operands as integers, the quotient given will be the algebraically next lower integer to the true quotient if the latter is not an integer. For example:

<table>
<thead>
<tr>
<th>Dividend</th>
<th>Divisor</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>10</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>199</td>
<td>10</td>
<td>19</td>
<td>9</td>
</tr>
<tr>
<td>-200</td>
<td>10</td>
<td>-20</td>
<td>0</td>
</tr>
<tr>
<td>-199</td>
<td>10</td>
<td>-20</td>
<td>1</td>
</tr>
<tr>
<td>-199</td>
<td>-10</td>
<td>19</td>
<td>-9</td>
</tr>
</tbody>
</table>
**DVR**

**Rounded Double-length Divide**

**Function**
Divide the double-length number in XX* by the single-length number in N(M), giving a rounded quotient.

**Format**

- **Operation Code**: DVR
- **Accumulator**: X
- **Operand**: N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to a store location which contains the divisor.

**Execution**
The double-length dividend in X and X* is divided by the single-length number in N(M). The single-length rounded quotient is left in X* after $2^{-24}$ has been added to effect the rounding. The double-length dividend and the single-length divisor may be integers, fractions or mixed numbers. (If the numbers have fractional parts, the number of bits after the binary point in the quotient is equal to the difference between the number of bits after the binary point in the dividend and divisor.) Negative numbers are catered for and a correctly signed quotient is produced. The contents of N(M) are unaltered. A remainder is left which satisfies the equation: Remainder = Dividend - Quotient x Divisor.

C is not used and will be left clear.

V will be set for the following conditions:

1. If the quotient exceeds the capacity of one computer word.
2. If an attempt is made to divide by zero, in which case the dividend is left unchanged in XX*.

**Modification**
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Example**
The single-length numbers are stored in NUMA and NUMB, with NUMA less than NUMB. It is required to divide NUMA by NUMB, and store the result in RES.
Machine Code Listing

045 X M N

Notes

1  DVR is to be preferred to DVD when fractions are used, because DVR produces the best single
- word approximation to the true quotient.

2  If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to
the symbol (not PLAN 1).

3  Some practical examples of the use of DVR are as follows:

<table>
<thead>
<tr>
<th>Dividend</th>
<th>Divisor</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>10</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>204</td>
<td>10</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>205</td>
<td>10</td>
<td>21</td>
<td>-5</td>
</tr>
<tr>
<td>-205</td>
<td>10</td>
<td>-20</td>
<td>-5</td>
</tr>
<tr>
<td>-205</td>
<td>-10</td>
<td>21</td>
<td>5</td>
</tr>
</tbody>
</table>
**DVS**

**Single-length Divide**

**Function**  
Divide the single-length number in X* by N(M)

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>XDV</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>N(M)</td>
<td></td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to the location which contains the divisor.

**Execution**

The single-length number in X* is divided by the single-length number in N(M). Although X is written in the accumulator field of the DVS statement, the dividend must be placed in X*: the initial content of X is irrelevant. The results of the operation are placed in X and X*, X holding the remainder and X* the unrounded quotient.

The single-length dividend and the single-length divisor may be integers, fractions or mixed numbers. (If the numbers have fractional parts, the number of bits after the binary point in the quotient is equal to the difference between the number of bits after the binary points in the dividend and divisor.) Negative numbers are catered for and a correctly signed quotient is produced. A remainder is left in X which satisfies the equation: Remainder = Dividend - Quotient x Divisor. The remainder in X will always have the same sign as the divisor. The contents of N(M) are unaltered.

C is not used and will be left clear.

Y will be set for the following conditions:

1. If the quotient exceeds the capacity of one computer word.
2. If an attempt is made to divide by zero, in which case the dividend is left unchanged in XX*.

**Modification**  
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Example**

Single-length integers are stored in NUMA and NUMB. Divide NUMA by NUMB. Place the quotient in QUOT and the remainder in REM.
### Machine Code Listing

046 X M N

### Notes

1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

2. Considering the operands as integers, the quotient given will be the algebraically next lower integer to the true quotient if the latter is not an integer. For example:

<table>
<thead>
<tr>
<th>Dividend</th>
<th>Divisor</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>10</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>199</td>
<td>10</td>
<td>19</td>
<td>9</td>
</tr>
<tr>
<td>-200</td>
<td>10</td>
<td>-20</td>
<td>0</td>
</tr>
<tr>
<td>-199</td>
<td>10</td>
<td>-20</td>
<td>1</td>
</tr>
<tr>
<td>-199</td>
<td>-10</td>
<td>19</td>
<td>-9</td>
</tr>
</tbody>
</table>
Logical EXCLUSIVE OR Direct Operand into X

Function
Perform a 'logical EXCLUSIVE OR' on the contents of X and the value of N(M) leaving the result in X.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>ERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the value to be used together with the contents of X in the 'logical EXCLUSIVE OR' operation.

Execution

A 'logical EXCLUSIVE OR' operation is performed with the contents of X and the value of N(M). Both words are regarded as 24-bit patterns rather than as numeric quantities. The ERN instruction places a 1-bit in X only where X and N(M) differ (i.e., a 1-bit is placed in X where there is a 1-bit in N(M) and a 0 in X or where there is a 0-bit in N(M) and a 1-bit in X). All other bits of X are set to 0. The contents of N(M) are unaltered.

If the operand is the symbolic name of a location previously defined under a #LOWER directive, the address of that location will be used as the appropriate bit pattern. If the operand is written as a symbolic identifier previously set by #DEFINE or #SET directives, the value that was assigned to that identifier is used as the appropriate bit pattern.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification

This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

1. If it is required to invert the state of a marker bit (say B15 of X4), this can be done as follows:

| LABEL | OPERATION | ACC | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1     | ERN       | #A00 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Chapter 4
2 Accumulator 3 contains either 0 or 4. Whichever it is, replace it by the other.

Machine Code Listing

122 X M N

Notes

1 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
ERS

Logical EXCLUSIVE OR into Store

Function
Perform a 'logical EXCLUSIVE OR' on the contents of N(M), and the contents of X, leaving
the result in N(M).

Format
Operation Code  ERS
Accumulator     X
Operand         N(M)

N may be:
1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
       decimal or octal integer.
2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the location whose contents are used in the
'logical EXCLUSIVE OR' operation.

Execution
A 'logical EXCLUSIVE OR' operation is performed on the contents of N(M) and the contents of X. Both
words are regarded as 24-bit binary patterns rather than arithmetic expressions. The ERS instruction
places a 1-bit in N(M) only where corresponding bits of N(M) and X are different. The contents of X
are unaltered.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M
are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken
as the operand.

Examples
1 Original contents of GRIPE: 001 111 101 110 001 101 100 101
   Original contents of X2: 101 110 000 100 010 011 010 111

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERS</td>
<td>GRIPE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation:

Result in GRIPE: 100 001 101 010 011 110 110 010
(X2 is unchanged.)
2 It is required to interchange two numbers, \( a \) in X4 and \( b \) in DUCKS. (This example illustrates the use of both ERX and ERS.)

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a)</td>
<td>ERX</td>
<td>DUCKS</td>
</tr>
<tr>
<td>(b)</td>
<td>ERS</td>
<td>DUCKS</td>
</tr>
<tr>
<td>(c)</td>
<td>ERX</td>
<td>DUCKS</td>
</tr>
</tbody>
</table>

(a) Original contents of X4: 011 101 110 001 010 011 111 000
   Original contents of DUCKS: 101 110 001 111 000 010 101 011
   Result in X4: 110 011 111 110 010 001 010 011

(b) Original contents of X4: 110 011 111 110 010 001 010 011
   Original contents of DUCKS: 101 110 001 111 000 010 101 011
   Result in DUCKS: 011 101 110 001 010 011 111 000

(c) Original contents of X4: 110 011 111 110 010 001 010 011
   Original contents of DUCKS: 011 101 110 001 010 011 111 000
   Result in X4: 101 110 001 111 000 010 101 011

**Machine Code Listing**

032 X M N

**Notes**

1 If operand type 1 (a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
Logical EXCLUSIVE OR into X

Function
Perform a 'logical EXCLUSIVE OR' on the contents of X and the contents of N(M), leaving
the result in X.

Format
Operation Code  ERX
Accumulator     X
Operand         N(M)

N may be:
1  A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.
2  An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range
       0 to 4095.
3  A literal (not PLAN 1).

The operand N(M) refers to the location the contents of which are used
 together with the contents of X in the 'logical EXCLUSIVE OR' operation.

Execution
A 'logical EXCLUSIVE OR' operation is performed on the contents of X and the contents of N(M). Both
words are regarded as 24-bit patterns rather than arithmetic expressions. The ERX instruction places
a 1-bit in X only where corresponding bits of X and N(M) are different. The contents of N(M) are
unaltered.

C  is not used and will be left clear.

V  is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M
are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken
as the operand.

Example
1  Original contents of X2:  110 011 000 111 101 001 100 100

   Original contents of MASK: 010 100 111 010 110 011 010 010

   Operation:
   ERX  2  MASK

   Result in X2:  100 111 111 101 011 010 110 110

(MASK is unchanged.)

Machine Code Listing
022  X  M  N

Notes
1  If operand type 1 (a) is an undefined symbol the compiler will allocate a location in lower data to
the symbol (not PLAN 1).
EXIT

PLAN 1,2,3,4

Function

Provide a re-entry to the calling routine after a subroutine has been performed.

Format

Operation Code  EXIT
Accumulator  X (link accumulator formed by the previous CALL instruction containing the link address).
Operand

N may be:

An absolute expression in the range 0 to 32,767, e.g.

(a) A decimal integer in the range 0 to 32,767.
(b) An octal integer in the range #0 to #7777.
(c) An absolute symbol with a value in the range 0 to 32,767.

The operand N specifies the value to be added to the link address in X to give the re-entry address.

Execution

The EXIT instruction causes control to be transferred to the location whose address is formed by adding the value specified by N to the link address contained in X. The link address is determined from X in accordance with the address mode and branch mode in which the program member is currently operating. If the program member is operating in compact mode and direct branch mode (15AM and DBM), bits 9 to 23 of X are taken as the link address. If the program member is operating in extended data mode (22AM), or in compact mode and extended branch mode (15AM and EBM), bits 2 to 23 of X are taken as the link address.

The address to which control will be transferred will thus be N words beyond the address held in accumulator X, therefore avoiding parameters (if any) held in locations following the CALL instruction.

Zero suppression mode is left set to the value stored in the link accumulator, X. If the program member is operating in compact mode and direct branch mode (15AM and DBM), bit 8 of X is taken to represent the value to which zero suppression mode is to be set. If the program member is operating in extended data mode (22AM), or in compact mode and extended branch mode (15AM and EBM), bit 1 of X is taken as representing the value to which zero suppression mode is to be set (but see note 6).

C is not used and will be left clear.

V is left set by the EXIT instruction if either V was left set by the previous instruction or bit 0 of X (which records the state of V before control was transferred by the CALL instruction) is set to 1, or if both these conditions apply; otherwise V is left unset.

Modification

The EXIT statement has no M-field.
Machine Code Listing

072 X N

Notes

1 See also the description of the CALL instruction.

2 EXIT is generally used to return control to an instruction one or more (depending on the value of N) beyond the previous CALL (or, at least, the CALL which set up the contents of X - not necessarily the last CALL instruction obeyed before the given EXIT).

3 EXIT may also be used to return control from an overlay unit called by an ENTER or a RECAL macro-instruction (see Chapter 5). The link accumulator used by the overlay package is always X1.

4 If the link accumulator is required for use between the CALL and the EXIT, or between the ENTER or RECAL and the EXIT, then its contents must be preserved in a storage location and loaded into the accumulator used by the EXIT instruction before the latter is obeyed.

5 If the EXIT instruction is preceded by a SMO instruction, the supplementary modifier is added to the sum of N and the link address to determine the address of the location to which control is to be transferred.

6 In determining the address of the location to which control is to be transferred, the sum of N, the link address and (if applicable) the supplementary modifier, is truncated to 15 bits if the program member is operating in compact mode and direct branch mode (15AM and DBM), or to 22 bits if the program member is operating in extended data mode (22AM) or in compact mode and extended branch mode (15AM and EBM).

   If the program member is operating in compact mode and extended branch mode (15AM and EBM), then N and (if applicable) the supplementary modifier are extended from 15 bits to 22 bits before the summation and subsequent truncation, by propagating the value of the most significant bit of the item. If the program member is operating in extended data mode (22AM), then N is extended to 22 bits, by propagating the value of the most significant bit, before the summation and subsequent truncation.

   If negative modification is employed in compact mode and direct branch mode (15AM and DBM), i.e., if the most significant bit of N or of the supplementary modifier is set to 1, then on some processors a carry will be propagated which will reverse the setting of zero suppression mode.

7 It is inadvisable to EXIT from a subroutine in a different address mode (or, if in compact mode throughout, in a different branch mode) from that in which it was entered, as the link address and zero suppression mode setting would then be liable to be misinterpreted.
FAD

Floating-point Add

**Function**
Add into floating-point accumulator A the floating-point number held in N(M) and N + 1(M). The answer is rounded and normalized.

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>FAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X = 0</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range 000 to 7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of two consecutive locations which contain the normalized floating-point number.

**Execution**
The FAD instruction causes the normalized floating-point number in N(M) and N + 1(M) to be added to the floating-point accumulator A. The contents of N(M) and N + 1(M) are left unchanged. The result is left in A.

**FOVR** will be set if overflow occurs.

**C** is not used and is left clear.

**V** is not used and remains unchanged.

**Modification** This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Machine Code Listing**

```
132  X  M  N
```

**Notes**

1. If operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data. (The symbol is given the value of the first location.)

2. X = 4 is also valid and will give a result to the same specification.
**Floating-point Divide**

**Function**
Divide the contents of the floating-point accumulator A by the floating-point in number N(M) and N + 1(M).

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>FDVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X = 0 The answer is rounded and normalized in A.</td>
</tr>
<tr>
<td></td>
<td>X = 4 The answer is rounded and normalized in A, and the function is inverted.</td>
</tr>
</tbody>
</table>

| Operand | N(M) |

N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of two consecutive locations which hold the normalized floating-point number.

**Execution**

If X = 0 is specified, then the contents of the floating-point accumulator A are divided by the normalized floating-point number in N(M) and N + 1(M), the result being left in A.

If X = 4 is specified, then the normalized floating-point number in N(M) and N + 1(M) is divided by the contents of A, the result being left in A.

In both cases, the contents of N(M) and N + 1(M) are left undisturbed.

**FOVR** will be set if overflow occurs.

**C** is not used and is left clear.

**V** is not used and remains unchanged.

**Modification**
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Examples**

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FDVD</td>
<td>0</td>
<td>HELP+3</td>
</tr>
<tr>
<td>2</td>
<td>FDVD</td>
<td>0</td>
<td>DATA</td>
</tr>
</tbody>
</table>

The first example above causes the contents of the floating-point accumulator A to be divided by the floating-point number in HELP + 3 and HELP + 4. The answer is rounded and normalized and left in A.

The second example causes the floating-point number in DATA and DATA + 1 to be divided by the contents of the floating-point accumulator A. The answer is rounded and normalized and left in A.
Machine Code Listing

135 X M N

Notes

1 If operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data. (The symbol is given the value of the first location.)
FIX

Convert Floating to Fixed

Function
Convert a floating-point number to fixed mid-point form.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Format</th>
<th>Accumulator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIX</td>
<td>Blank</td>
<td>N(M)</td>
<td></td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of two consecutive locations in which
the mid-point number is to be stored.

Execution

The FIX instruction denormalizes the contents of floating-point accumulator A (i.e. it adjusts the
exponent to 279) and alters the argument accordingly. The resulting number is stored in mid-point
form in N(M) and N + 1(M), the contents of floating-point accumulator A being undisturbed.

FOVR is used and remains unchanged. (See V below.)

C is not used and will be left clear.

V will be set if
1. the result is above the standard mid-point range, or
2. exponent overflow was indicated in the floating-point accumulator A.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M
are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken
as the operand.

Machine Code Listing

131 0 M N

Notes

1. If operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations
   in lower data. (The symbol is given the value of the first location.)
FLOAT

PLAN 1,2,3

Convert Fixed to Floating-point

Function
Convert a mid-point number to floating-point form.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>N(M)</td>
</tr>
<tr>
<td>Operand</td>
<td></td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
textual form.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of two consecutive locations which
hold the fixed-point number.

Execution

The FLOAT instruction normalizes the fixed mid-point number held in N(M) and N + 1(M) with respect
to an exponent of 279, and loads the result into the floating-point accumulator A. The contents of N(M)
and N + 1(M) are unaltered.

C is not used and will be left clear.

Y is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are
taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the
operand.

Machine Code Listing

130 0 M N

Notes

1. If operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in
   lower data. (The symbol is given the value of the first location.)
Floating-point Multiply

Function
Multiply the contents of the floating-point accumulator A by the floating-point number held in N(M) and N + 1(M). The answer is rounded and normalized.

Format
- Operation Code: FMPY
- Accumulator: X = 0
- Operand: N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of two consecutive locations containing a normalized floating-point number.

Execution
The FMPY instruction causes the contents of the floating-point accumulator A to be multiplied by the normalized floating-point number in N(M) and N + 1(M). The contents of N(M) and N + 1(M) are unaltered. The result is left in A.

FVOY will be set if exponent overflow occurred.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing
134 X M N

Notes
1. If operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data. (The symbol will be given the value of the first location.)
2. X = 4 is also valid and will give the same result.
PLAN 1,2,3

Function
Subtract from the contents of the floating-point accumulator A the floating-point number held in N(M) and N + 1(M).

Format
Operation Code  FSB
Accumulator
X = 0 The answer is rounded and normalized.
X = 4 The answer is rounded and normalized, and the operation is performed with A and N interchanged.
Operand N(M)

If X is specified, N may be:
1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of two consecutive locations which contain the normalized floating-point number.

Execution
If X = 0 is specified, the normalized floating-point number in N(M) and N + 1(M) is subtracted from the contents of the floating-point accumulator A, the result being left in A.

If X = 4 is specified, the contents of the floating-point accumulator are subtracted from the floating-point number in N(M) and N + 1(M), the result being left in A. In both cases the contents of N(M) and N + 1(M) are left unaltered.

FOVR will be set if exponent overflow occurs.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Examples

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC 15 14</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FSB</td>
<td>0. KNOK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FSB</td>
<td>0. WOLF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first example above causes the normalized floating-point number in KNOK and KNOK + 1 to be subtracted from the floating-point accumulator A.

The answer is rounded and normalized and left in A.
The second example causes the contents of the floating-point accumulator A to be subtracted from the normalized floating-point number in WOLDE and WOLDE + 1. The answer is rounded and normalized, and left in A.

**Machine Code Listing**

133 X M N

**Notes**

1 If operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data. (The symbol is given the value of the first location.)
PLAN 3,4
(Magnetic Tape Macro-instruction)

Function
Move a magnetic tape forward to the next tape mark.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>FTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>A decimal digit in the range 0 to 7, specifying the program's unit number for the magnetic tape unit.</td>
</tr>
<tr>
<td>Operand</td>
<td>Blank</td>
</tr>
</tbody>
</table>

Execution
FTM causes the tape on the specified unit to be wound on until the next tape mark is detected. The tape will stop in a position which will enable the block that follows the tape mark (i.e. the qualifier block) to be read.

Modification
The FTM macro-instruction has no M-field.

Machine Code Listing

```
157 X 0 n LT
```

where \( n \) is the relative address in the literal table of the first of two consecutive locations which hold a control area of form:

<table>
<thead>
<tr>
<th>first word</th>
<th>second word</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/#4</td>
<td>0</td>
</tr>
</tbody>
</table>

Notes

1. This is one of a set of magnetic tape macro-instructions that permit the execution of simple basic functions on magnetic tape without requiring the programmer to set up control areas. (See also BSP, BTM, CLOSE, REW, SCR, UNL, WTM.)
GIVE

Give Information

Function
Cause information to be put into X or XX* according to the value of N(M). With some values of N(M), additionally request changes in the internal operating environment.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>GIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
An absolute expression in the range 0 to 4095, e.g.
(a) A decimal integer in the range 0 to 4095,
(b) An octal integer in the range #0 to #7777.
(c) A previously defined absolute symbol with a value in the range 0 to 4095.

Execution

The GIVE instruction causes information to be put into accumulator X or two consecutive accumulators XX*, according to the value of N(M). With some values of N(M), changes may be requested in the internal operating environment, depending on the initial contents of X; the final contents of X in such cases indicate the extent to which the request has been implemented.

N(M) = 0
Give today's date in binary in X.
Accumulator X will contain in binary form the number of days from 1st January 1900, that day being day 1.

N(M) = 1
Give today's date in character form in XX*.
Accumulators XX* will contain eight characters in the form DD (day)/MM (month)/YY (year).

N(M) = 2
Give time in character form in XX*.
Accumulators XX* will contain eight characters in the form HH (hours) /MM (minutes) /SS (seconds).
This time is on a 24-hour basis, midnight being given by 00 /00 /00.
The time is available only in configurations with equipment suitable for determining real time. If the instruction is given and no such equipment is available in the configuration, the XX* are cleared.

N(M) = 3
Give the total number of words of core store allocated to this program as a binary integer in X.
Accumulator X will contain an integer equal to the number of core store locations allocated to the program.

If bit 0 of X is zero, the program is dense.
If bit 0 of X is set, the program is sparse.

N(M) = 4
Request specified amount of store.
The initial contents of X are interpreted as a request to the operating environment as follows:
If bit 0 of X is zero, the program is dense
If bit 0 of X is set, the program is sparse

Bits 2 to 23 contain a number indicating the number of words of store to be assigned to the program.
In a paged environment, this instruction can be used to make a dense program sparse or vice versa.
The operating environment rounds the request to the next highest multiple of store increment. The minimum valid store request is 64 where there are no subprograms or only one, and 128 where there are two or more subprograms. The final contents of X indicate whether the program is sparse or dense and the number of words of store that are assigned to the program when the instruction has been performed.
If the instruction causes additional store to be assigned to the program, that additional store will contain zero in all words, except possibly where a GIVE with \( N(M) = 12 \) instruction has been performed.

The instruction may cause other programs in the system to be suspended for a significant period of time. A subprogram issuing the instruction is liable to be suspended indefinitely. It is assumed that such programs are loaded to occupy the first program slots.

The program store size in the program request block may be updated when an alternation in the amount of store assigned occurs.

\( N(M) = 5 \)

Give details of Executive and Central Processor.

The reply in accumulators X and X* gives details of features, not common throughout the series, which are present in this processor, excluding any information about peripheral devices. For details of reply information see Note 3.

The initial contents of X and X* are overwritten and are of no significance.

\( N(M) = 8 \)

Give current address mode and branch mode of issuing member.

Accumulator X will contain an indication of the address mode and branch mode in which the issuing program member is currently operating, as follows:

- **Bits 0 to 20**: Undefined.
- **Bit 21**: Set to zero if in direct branch mode. Set to one if in extended branch mode.
- **Bit 22**: Undefined.
- **Bit 23**: Set to zero if in 15-bit address mode. Set to one if in 22-bit address mode.

\( N(M) = 9 \)

Request a change in address mode and/or branch mode of issuing member.

Accumulator X must contain an indication of the address mode and branch mode in which the issuing program member requires to operate, as follows:

- **Bits 0 to 20**: Zero
- **Bit 21**: Zero if direct branch mode is required. One if extended branch mode is required.
- **Bit 22**: Zero
- **Bit 23**: Zero if 15-bit address mode is required. One if 22-bit address mode is required.

On completion of the instruction, reply information is put into accumulator X indicating the address mode and branch mode in which the program member is operating. The format of the reply information is the same as for \( N(M) = 8 \).

In environments which do not offer a choice of address modes or of branch modes, an attempt to switch to a non-available mode will leave the issuing member in the mode in which it was already operating, with the reply information set accordingly. In this respect the address mode request and the branch mode request operate independently of each other.

\( N(M) = 10 \)

Give relative program mill time in micro seconds.

Accumulators XX* will contain the approximate mill time in micro seconds that a program has used in a given time. The mill time given depends not only on the operating environment but on other programs running concurrently. In particular, peripheral transfer hesitation time may be attributed to any current program.

The instruction is effective only where the operating environment has a mill timer. In other environments the instruction is null.

\( N(M) = 11 \)

Give the time in seconds since midnight as a mid-point number.
Accumulators XX* will contain a standard mid-point number (with X as the integral part and X* the fractional part) giving the time in seconds since midnight.

In operating environments which do not implement this instruction, the instruction clears X and leaves the contents of X* unchanged.

N(M) = 12

The instruction allows the operating environment to increase its efficiency by making use of the fact that only the active part of the program store image need be in store while the program is being obeyed. The instruction may be implemented in unboxed operating environments where active programs are liable to be interchanged between processor store and backing store. In paged environments such efficiency can be achieved in other ways.

The active store limit of a program is the limit beyond which a program will not refer until after a GIVE with N(M) = 4 or 12 instruction. The dormant limit of a program is the limit of the store image area assigned to the program. The dormant limit is greater than or equal to the active limit. The area between the active and dormant limits is the dormant area. Except where this instruction has been obeyed, the values of the active and dormant limits are equal.

The initial contents of X* are interpreted as the active store limit. If the contents of X are less than the initial dormant limit, the instruction rounds the initial contents of X to the next highest multiple of store increment, and sets the active store limit equal to the rounded contents of X.

Any additional active store is set to the value it had before it became dormant. If the initial contents of X are equal to or greater than the dormant limit, the active limit is set equal to the dormant limit.

If the instruction is performed and the dormant limit is greater than the active store limit, a subsequent GIVE with N(M) = 4 instruction will set any additional store below the dormant limit to its original state before this instruction was issued and any additional store above the dormant limit to zero. After the GIVE N(M) = 4 instruction has been performed, the active and dormant store limits are set equal and given as a reply value in X.

In environments which do not implement this instruction, the reply in X is set to zero and the active and dormant limits which are always equal are unchanged.

In paged environments this instruction is identical to GIVE N(M) = 3. Thus the active and dormant limits must be equal and are unchanged by the instruction.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Examples

| LABEL | OPERATION | ACC | 12 11 10 09 08 07 06 05 04 03 02 01 00 | OPERAND | 12 11 10 09 08 07 06 05 04 03 02 01 00 |
|-------|-----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1     | GIVE      | 4   | 0   |     |     |     |     |     |     |     |     |     |     |     |
| 2     | GIVE      | 7   | 1   |     |     |     |     |     |     |     |     |     |     |     |
| 3     | GIVE      | 5   | 2   |     |     |     |     |     |     |     |     |     |     |     |
| 4     | GIVE      | 2   | 3   |     |     |     |     |     |     |     |     |     |     |     |

5 Try to increase store allocation up to 8096.

| LABEL | OPERATION | ACC | 12 11 10 09 08 07 06 05 04 03 02 01 00 | OPERAND | 12 11 10 09 08 07 06 05 04 03 02 01 00 |
|-------|-----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1     | LDX       | 5   | '8096' |     |     |     |     |     |     |     |     |     |     |     |     |
| 2     | STO       | 5   | REQUEST |     |     |     |     |     |     |     |     |     |     |     |     |
| 3     | GIVE      | 5   | 4   |     |     |     |     |     |     |     |     |     |     |     |     |
| 4     | BXGE      | 5   | REQUEST, VM +2 |     |     |     |     |     |     |     |     |     |     |     |     |
| 5     | SUSWT     | 2   | HNG  |     |     |     |     |     |     |     |     |     |     |     |     |
Machine Code Listing

165 X M N

Notes

1 A GIVE instruction with any value of N(M) not defined above nor in the range 16 to 19 (see below)
clears C, but is otherwise undefined.

2 If N(M) = 2 and the only means of determining real time is some device attached as part of a complex of
communications equipment, e.g. via a multiplexor, then this variant of the instruction may not be effectively
implemented.

3 Where N(M) = 5 the basic reply to the GIVE instruction is a 48-bit binary pattern, generally bit-significant.
If a bit is set to 1 it indicates that a particular feature is present; if the bit is set to 0, that feature is not
present. Bit 0 of X has a special significance, in that if it is set to 1, then the contents of X* are meaningful;
whereas if it is set to 0 the contents of X* are indeterminate, but the features which otherwise would be
indicated in X* are all absent. (This is to minimize the amount of executive coding on small processors.)
The significances of individual bits are as follows:

Bit: When set to 1, means:
0 X* has meaningful contents. (See above.)
1 The configuration has a console typewriter.
2 Undefined.
3 Extended data mode and extended branch mode are available. (Bit 4 will also be 1.)
4 BCT, MVCH and SMO instructions are available. Double-length shifts are performed by hardware.
5 NORM single-length and double-length instructions are available by hardware, irrespective of the
availability of other floating-point instructions. Double-length shifts are performed by hardware.
If bit 3 is also set, MVCH will also be performed by hardware.
6 Floating-point instructions are available. If bit 4 is also set, the BFP instruction will also be
available.
7 FAD, FSB, FMPY and FDVD instructions will accept operands in normalized or unnormalized
form, and the X field in these instructions may be set to produce unrounded or unnormalized
results.
8 Subprogramming is available.
9 The original subprogramming system is available, in addition to the current one.
10 Undefined.
11 Undefined.
12 Multiplication and division are performed by hardware.
13 CDB and CBD instructions are performed by hardware.
14 MOVE and SUM instructions are performed by hardware.
15 Floating-point instructions (except in some processors FLOAT and FIX) are performed by
hardware.
16 to 23 Undefined.
The significances of the bits in X* are at present undefined.

4 If N(M) = 9, and an address mode request or a branch mode request fails because of environmental limitations,
then the issuing program member must take appropriate action. A program member may request a change
which would not alter either mode from those already established.
Additional Facilities in Paged Environments

GIVE instructions with N(M) having a value in the range 16 to 19 are applicable to paged operating environments only. The instructions allow a sparse or dense program to determine the status of a specified area within its virtual store and to change the status of non-shareable areas. The area is specified in accumulators X and X* and is defined by giving the page address (as a multiple of 1024 words) of the first page and the number of pages which comprise the area. The status of each page of an area is controlled by software status bits known as permission (execute, read and write), shareable and used bits. A page is considered to have been used if it was written to when the binary program was loaded or it has subsequently been accessed by the program or a job description command.

N(M) = 16 Report the status of an area.

Accumulators X and X* must initially contain the following information:

X Bits 0 to 4 Set to 1 where a report is requested on particular status bits as follows:
  Bit 0 = execute status
  Bit 1 = read status
  Bit 2 = write status
  Bit 3 = shareable status
  Bit 4 = used status

Bits 5 to 11 Zero

Bits 12 to 23 Page address of the start of the area

X* Bits 0 to 10 Undefined

Bits 11 to 23 Number of pages in the requested area

Each page of the requested area is matched in turn with the first page of the area requested in X, until a page is found having the requested statuses differing from those in the first page of the area. The instruction gives a reply in X and X*, reporting on the area covered until a mismatch occurred.

The reply will be given in X and X* indicating the statuses as follows:

X Bit 0 = 1 At least one page in the reported area does not have execute permission
  Bit 1 = 1 At least one page in the reported area does not have read permission
  Bit 2 = 1 At least one page in the reported area does not have write permission
  Bit 3 = 1 At least one page in the reported area is shareable
  Bit 4 = 1 At least one page in the reported area has been used

Bits 10 to 23 The page address of the start of the reported area

X* Bit 0 = 1 All pages in the reported area do not have execute permission
  Bit 1 = 1 All pages in the reported area do not have read permission
  Bit 2 = 1 All pages in the reported area do not have write permission
  Bit 3 = 1 All pages in the reported area are shareable
  Bit 4 = 1 All pages in the reported area have been used

Bits 11 to 23 Number of pages in the reported area, that is, until a mismatch occurred

Notes

1 Where a report is requested on particular status bits, that is, where any of bits 0 to 4 of X were initially set = 1, and are given as 1 in the reply, it follows that all the pages in the reported area have that status.

2 Where a report is not requested on particular status bits, that is, where any of bits 0 to 4 of X were initially set = 0, a report will be given for those status bits, but the meaning in the reply will be different to that given for the status bits which were to be specifically reported on. The reply will give a report of the whole of the requested area, since there is no requested status to be matched through the area.
N(M) = 17
Change the specified status.
Accumulators X and X* will initially contain the following information:

X
Bit 0 = 0 Give execute permission
    = 1 Withhold execute permission
Bit 1 = 0 Give read permission
    = 1 Withhold read permission
Bit 2 = 0 Give write permission
    = 1 Withhold write permission

Bits 3 to 11 Zero
Bits 12 to 23 Page address of the start of the area

X*
Bits 0 to 10 Zero
Bits 11 to 23 Number of pages in the area

It is not possible to alter the shareable or used status of an area using this instruction. Bits 3 and 4 of X must therefore initially be zero. If any page in the area has shareable status, the result of the instruction is undefined.

If all three permissions are withheld from an area, any attempt to access that area will subsequently be treated as an error.

The reply will be given as follows:

X
Bits 0 to 4 These bits will be set = 1 if at least one page in the requested area has the indicated status (see GIVE N(M) = 16)
Bits 5 to 23 The setting of these bits will be that given as the initial contents of X

X*
Bits 0 to 4 These bits will be set if all the pages in the requested area have the indicated status (see GIVE N(M) = 16)
Bits 5 to 23 The setting of these bits will be that given as the initial contents of X*

N(M) = 18
Give the specified permission
Accumulators X and X* will initially contain the following information:

X
Bit 0 = 1 Give execute permission
Bit 1 = 1 Give read permission
Bit 2 = 1 Give write permission
Bits 3 to 11 Zero

Bits 12 to 23 Page address of the start of the area

X*
Bits 0 to 10 Zero

Bits 11 to 23 Number of pages in the area

It is not possible to alter the shareable or used status of an area using this instruction. Bits 3 and 4 of X must therefore initially be zero. If any page in the area has shareable status, the result of the instruction is undefined.

The reply will be given as follows:

X
Bits 0 to 2 Any of these bits that were set = 1 initially in X will be set = 1
Where any of these bits were initially set = 0, the reply in bits 0 to 2 will be set if at least one page in the requested area has the indicated status

Bits 3 to 23 The setting of these bits will be that given as the initial contents of X

X*
Bits 0 to 2 The setting of these bits will be that of bits 0 to 2 of the initial contents of X, thus indicating that the whole area has the indicated status

Bits 3 to 23 The setting of these bits will be that given as the initial contents of X*
N(M) = 19  Withhold the specified permission.

Accumulators X and X* will initially contain the following information:

X  
- Bits 0 = 1  Withhold execute permission
- Bit 1 = 1  Withhold read permission
- Bit 2 = 1  Withhold write permission
- Bits 3 to 11  Zero
- Bits 12 to 23  Page address of the start of the area

X*  
- Bits 0 to 10  Zero
- Bits 11 to 23  Number of pages in the area

It is not possible to alter the shareable or used status of an area using this instruction. Bits 3 and 4 must therefore initially be zero. If any page in the area has shareable status, the result of the instruction is undefined.

The reply will be given as follows:

X  
- Bits 0 to 2  Any of these bits that were set = 1 initially in X will be set = 1. Where any of these bits were initially set = 0, the reply in bits 0 to 2 will be set if at least one page in the requested area has the indicated status
- Bits 3 to 23  The setting of these bits will be that given as the initial contents of X

X*  
- Bits 0 to 2  The setting of these bits will be that of Bits 0 to 2 of the initial contents of X, thus indicating that the whole area has the indicated status
- Bits 3 to 23  The setting of these bits will be that given as the initial contents of X*

Notes

In general, the area specified in the instruction with N(M) in the range 17 to 19 should consist of a whole number of quires. Then each page in the quire, used or not, will be given the requested permission (unless the page is shareable). If the area specified does not consist of a whole number of quires, then pages within the quires that are only partly contained in the area will only have their permission bits altered if they have been used and are not shareable. Pages within quires that are completely contained in the area will always have their permission bits altered.

It is illegal for the instruction with N(M) in the range 17 to 19 to refer to page 0 of a program.
LAEZ

Function
Clear the floating-point accumulator extension.

Format
\begin{align*}
\text{Operation Code} & : \quad \text{LAEZ} \\
\text{Accumulator} & : \quad \text{Blank} \\
\text{Operand} & : \quad \text{Blank}
\end{align*}

Execution
The LAEZ instruction clears the floating-point accumulator extension.

\begin{itemize}
\item \texttt{F0VR} will remain unchanged.
\item \texttt{C} is not used and will be left clear.
\item \texttt{V} will remain unchanged.
\end{itemize}

Modification
This statement has no M-field.

Machine Code Listing
\begin{align*}
136 & \quad 3 \quad 0 \quad 0
\end{align*}
LDCH

Load Character to X

Function: Load a character from N(M) and place it in X.

Format:
- **Operation Code**: LDCH
- **Accumulator**: X
- **Operand**: N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.
3. A literal (not PLAN 1).

The operand N(M) refers to the location from which the character is to be loaded.

Execution

In its unmodified form the LDCH instruction will make bits 18 to 23 (the least significant character) of X equal to bits 18 to 23 of N.

Bits 0 to 17 of X are cleared. N is left unchanged.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification: The LDCH instruction has an M-field. When modified by an accumulator whose contents take the form of a character index word, the instruction extracts the character at the character position (n0 to n3) specified by bits 0 and 1 of the index word, from the word specified by the sum of N and bits 9 to 23 of the index word and loads it into bits 18 to 23 of X. In extended data mode, the least significant 22 bits of N + M are taken as the operand. B0 to B17 are cleared and N(M) is left unchanged.

Examples:

1. X1 holds a character index word: 0/3.2

   X6 holds:
   - 011 010 110 101 001 100 110 100
   - INF + 3 holds:
   - 001 101 010 111 101 010 101 110

   Operation:
   - LDCH 6
   - INF(1)

   Result in X6:
   - 000 000 000 000 000 000 101 010

   This example has caused the character in the n2 position of INF + 3 to be loaded into the n3 position of X6.
2. It is required to find the sum of 37 single-digit numbers held in character form in core store. The first character is stored in ANTY + 1.2 in upper data and the next 36 in the succeeding 36 character positions. The result is to be stored in RES in lower data.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ST0Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDK</td>
<td>1</td>
<td>'31, (ANTY+1; -2'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDCH</td>
<td>4</td>
<td>0(1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADS</td>
<td>4</td>
<td>RES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCHX</td>
<td>1</td>
<td>4-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first instruction clears RES.

The second instruction leads a character index word into X1 to control the extraction of 37 successive characters from the store. (As literals are not available in PLAN 1, the index word in that case would have been defined as an item of constant data.) The third instruction extracts a character from store and loads it into X4. The fourth instruction adds the contents of X4 into RES. The fifth instruction advances the character index word by one character position and decreases the count by 1, thus ensuring that successive characters are extracted from store each time the instruction is obeyed.

**Machine Code Listing**

024 X M N

**Notes**

1. Though a character may be extracted from any character position of N(M) by the modified form of the LDCH statement, the character is always loaded into the n3 position of X. Note too, that an LDCH statement with a zero modifier has a different effect from an unmodified LDCH. This is true whether or not there is a supplementary modifier; supplementary modification by a SMO instruction does not affect the character position specified.

2. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
LDCM

Load Counter/Modifier

Function
Load the contents of N(M) and N+1(M) into two accumulators.

Format
Operation Code
LDCM

Accumulator
XcXm

Xc and Xm need not be adjacent accumulators.

Operand
N(M)

N may be:
1  A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name which refers to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.
2  An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range
       0 to 4095.

The operand N(M) refers to the first to two consecutive locations the
contents of which are to be loaded into Xc and Xm.

Execution
The LDCM macro-instruction will generate two machine code instructions. The contents of N(M) are
loaded into Xc, and the contents of N+1(M) are loaded into Xm. If C was left set by the previous instruc-
tion, 1 is added to Xm.

C  is used and will be left clear.

Y  will be set if N+1(M) contains #37777777 and C was set when the LDCM instruction came to be
    obeyed.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N+M are
taken as the operand if the program is operating in compact mode; or the least significant 22 bits of
N+M are taken as the operand if the program is operating in extended data mode.

Machine Code Listing
000  Xm  M  N+1
000  Xc  M  N

Notes
1  LDCM  XcXm  N(M)
   is equivalent to
   LDX  Xm  N+1(M)
   LDX  Xc  N(M)
2  If operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in
   lower data (the symbol is given the value of the first location).
The LDVM macro-instruction is intended for use in extended data mode programs, in conjunction with the PLAN 4 macro-instructions BUX, BDX and BCHX. It provides a convenient means of loading the two locations which are treated as a counter/modifier by those macro-instructions.
LDCT

Load Count

Function
Set a count of value N(M) in the count position of X. The remaining bits of X are cleared.

Format
- Operation Code: LDCT
- Accumulator: X
- Operand: N(M)

N may be:
- An absolute expression in the range 0 to 4095, e.g.
- A decimal integer in the range 0 to 4095.
- An octal integer in the range #0 to #7777.
- A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) specifies the value of the count to be set which should be in the range 0 to 511 (inclusive).

Execution
The LDCT instruction sets a count in bits 0 to 8 of X equal to the value specified by the least significant nine bits of N(M). The remainder of X (bits 9 to 23) is cleared.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
The LDCT statement has an M-field. When modified, the count set will be the least significant nine bits of N + M.

Machine Code Listing
124 X M N

Notes
1 This operation may be used to set the count part of a character index word. It should be remembered however (see BCHX) that the maximum count which may be held in such a word is 127 (in bits 2 to 8). Any value given in N of a LDCT operation which exceeds 127 will therefore affect the contents of bits 0 and 1 which are used as a character pointer. Both the count and the character pointer may be set as required by giving a LDCT instruction with the appropriate value in N.

? After a count has been set in bits 0 to 8 of X by a LDCT operation it may be required also to set a value in bits 9 to 23, so that X may be used as a modifier. Such a value may be loaded by a subsequent ADX or ORX instruction.
LDEX

Load Exponent

Function
Load into X bits 15 to 23 of the contents of N(M).

Format
- **Operation Code**: LDEX
- **Accumulator**: X
- **Operand**: N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to the location containing the nine bits which are to be loaded in X.

Execution

The LDEX instruction causes bits 15 to 23 of X to be made equal to bits 15 to 23 of N(M). Bits 0 to 14 of X are cleared and N(M) is left unchanged.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

Original contents of X5: 011 001 111 001 111 000 010 100

Original contents of DEPT: 100 010 101 101 111 110 010

| LABEL | OPERATION | ACC. | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------|-----------|------|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
|       | LDEX      |      |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
|       | DEPT      |      |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |

Operation:

Results in X5:
000 000 000 000 111 110 010

Machine Code Listing

025 X M N
Notes

1. LDEX is used mainly for handling the floating-point exponent, but can be used for unpacking any other type of data.

2. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
**PLAN 3**  
(Macro-instruction)

**Function**  
Load into X bits 9 to 23 of the contents of N(M).

**Format**  
*Operation Code*  
LDLA  

*Accumulator*  
X  

*Operand*  
N(M)  

N may be:

1. A relative expression which refers to a lower data location, e.g.  
   (a) A symbolic name referring to a lower data location.  
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.  
   (a) A decimal integer in the range 0 to 4095.  
   (b) An octal integer in the range #0 to #7777.  
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal.

The operand N(M) refers to the location containing the 15 bits which are to be loaded into X.

**Execution**  
The LDLA instruction causes bits 9 to 23 of X to be made equal to bits 9 to 23 of the contents of N(M).  
If C was left set by the previous instruction, 1 will be added into X. Bits 0 to 8 of X are cleared and  
N(M) is left unchanged.

C is used (see above) and will be left clear.

V is not used and remains unchanged.

**Modification**  
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Example**  
It is required to load the least significant bits of GROSS into the least significant 15 bits of XI.

```
| LABEL | OPERATION | ACC | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 15 | 32 | 64 | 128 | 256 | 512 | 1024 |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| LDLA  | GROSS     |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
```

The macro-instruction will cause the following series of instructions to be generated in machine code.

```
| LABEL | OPERATION | ACC | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 15 | 32 | 64 | 128 | 256 | 512 | 1024 |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| LDX   | GROSS     |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |
| ADDX  | #777777    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |
```

432217721  

Chapter 4
Machine Code Listing
000 X M N
020 X 0 n LT

where $n$ is the relative address in the literal table of a word containing the value of #7777.

Notes
1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol.
**PLAN 1,2,3**

**Function**
Load into accumulator X the value quoted in N(M).

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Accumulator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDN</td>
<td>X</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the value to be loaded into X.

**Execution**

The LDN instruction loads the value of N(M) into X. If C was left set by the previous instruction, 1 will be added to the result in X. The contents of N(M) are unaltered.

If the operand is the symbolic name of a location previously defined under a #LOWER directive, the address of that location will be loaded into X. If the operand is written as a symbolic identifier previously set by #DEFINE or #SET directives, the value that was assigned to that identifier is loaded into X.

C is used and will be left clear.

V is not used and remains unchanged.

**Modification**
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Machine Code Listing**

100 X M N

**Notes**

1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
LDNC

Load Direct Operand into X with Carry

Function
Load into X the value quoted in the operand field, setting C if carry occurs.

Format
Operation Code       LDNC
Accumulator          X
Operand               N(M)

N may be:
1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
do decimal or octal integer.
2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the value to be loaded into X.

Execution
The LDNC instruction is similar to the LDN instruction. The LDNC instruction loads the value of N(M)
into X. If C was left set by the previous instruction, 1 will be added to the result in X. The contents
of N(M) are unaltered.

At the conclusion of this instruction, B0 of the result will always be zero. If the operand is the symbolic
name of a location previously defined under a #LOWER directive, then the address of that location will
be loaded into X. If the operand is written as a symbolic identifier previously set by #DEFINE or #SET
directives, the value that was assigned to that identifier is loaded into X.

C is used (but cannot be set) and will be left clear (see Note 1).

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are
taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the
operand.

Machine Code Listing

104 X M N

Notes
1 The LDNC statement has the same effect as the LDN statement since carry cannot occur.
2 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to
   the symbol (not PLAN 1).
PLAN 2,3,4
(Macro-instruction)

Function
Load accumulator X with the value quoted in the operand field; this must be a label in the program area.

Format
Operation Code  LDPL
Accumulator     X
Operand         N

N may be:
1 A relative expression which refers to a program instruction, e.g.
   (a) A symbolic name that is the label of a program instruction.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2 A relative operand.

The operand N refers to the program location whose address is to be loaded into X.

Execution
The LDPL instruction causes the 15 bits, 9 to 23, of the address of the program location quoted in N to be loaded into X. If C was left set by the previous instruction, 1 will be added into the result in X. The contents of N are unaltered.

C is used (see above) and will be left clear.
V is not used and remains unchanged.

Modification The LDPL statement has no M-field.

Examples

|   | LABEL | OPERATION | ACC  | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
|---|-------|-----------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1 |       | LDPL      |      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 2 |       |           |      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Machine Code Listing
000 X 0 n LT

where word n of the literals area contains the operand N.

Notes
1 The main purpose of this instruction is to provide a convenient way of loading a program address to a modification register for use as a modifier.
2 In PLAN 4, this instruction causes a 22-bit address to be loaded into X if the segment is in 22AM.
LDSA

Load Short Address

Function Load into X bits 12 to 23 of the contents of N(M).

Format

Operation Code LDSA
Accumulator X
Operand N(M)

If X is specified, N may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
       decimal or octal integer.

2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

3 A literal.

The operand N(M) refers to the location containing the 12 bits that are to be
   loaded into X.

Execution

The LDSA instruction causes the least significant 12 bits (bits 12 to 23) of X to be made equal to bits 12 to 23
of the contents of N(M). Bits 0 to 11 of X are cleared and N(M) is left unchanged. If C was left set by the previous
instruction, I will be added into X.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as
the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

It is required to load the least significant 12 bits of CASH into the least significant 12 bits of XI.

The macro-instruction will cause the following series of instructions to be generated in machine code.

132 Chapter 4
Machine Code Listing

000 X M N
120 X 0  #7777 or 4095

Notes

1 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol.
LDX

Load to X

Function
Load the contents of N(M) into X.
Load the contents of N(M) and N + 1(M) into XX*.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>X or XX*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>N(M)</td>
</tr>
<tr>
<td>Operand</td>
<td></td>
</tr>
</tbody>
</table>

If X is specified, N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range
0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to the location the contents of which are to be
loaded into X.

If XX* is specified, N may be types 1 and 2 above. When N is type 2 it should
be in the range 0 to 4094. The operand N(M) refers to the first of two
consecutive locations the contents of which are to be loaded into XX*.

Execution

If X is specified, the contents of N(M) are loaded into X. If C was left set by the previous instruction, 1 is added
to the result in X. The contents of N(M) are unaltered.

If XX* is specified, the contents of N(M) and N + 1(M) are loaded into XX*, and 1 is added to X* if C was left
set by the previous instruction. The contents of N(M) and N + 1(M) are unaltered.

C is used and will be left clear.

V will be set if N(M) (single-length) or N + 1(M) (double-length) contains #37777777 and C was set when the
LDX instruction came to be obeyed.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as
the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing

When X is specified: 000 X M N
When XX* is specified: 000 X* M N + 1
                     000 X M N
Notes

1 If X is specified and operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

If XX* is specified and operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data (the symbol is given the value of the first location). Not PLAN 1.

2 LDX XX* N(M)
   is equivalent to
   LDX X* N + 1(M)
   LDX X N(M)
LDXC

Load into X with Carry

Function
Load the contents of N(M) into X, setting C if carry occurs.

Format
<table>
<thead>
<tr>
<th>Operation Code</th>
<th>LDXC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #00 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to the location the contents of which are to be loaded into X.

Execution
The contents of N(M) are loaded into X and if C was left set by the previous instruction, 1 is added to the result in X. The contents of N(M) are unaltered. At the conclusion of this instruction, B0 of X will always be zero.

C is used and will be set if carry occurs. See Chapter 1, page 6.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example
In the following example, the single-length positive quantity in ADJ is added to the triple-length number in TAX, TAX + 1 and TAX + 2, leaving the result in X5, X6 and X7.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>93</th>
<th>94</th>
<th>95</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX</td>
<td>7. ADJ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADAC</td>
<td>7. TAX+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDXC</td>
<td>6. TAX+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX</td>
<td>5. TAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Machine Code Listing

004 X M N
Notes

1. The LDXC instruction should be used for loading the less significant words of multi-length numbers. C will then be added correctly.

2. If used for non-numeric data; or for the most significant word of a number, an incorrect result may be obtained since a 1 in the B0 position will cause C to be set and it will not be stored correctly.

3. If operand type J(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
**Load Extended Floating-point**

**Function**  
Load the extended floating-point number in N(M) to N + 3(M) in the extended floating-point accumulator.

**Format**
- **Operating Code**: LEFP
- **Accumulator**: Blank
- **Operand**: N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of four consecutive locations that hold the extended floating-point number.

**Execution**
The LEFP instruction loads the extended floating-point accumulator with the number N(M) to N + 3(M). The contents of N(M) to N + 3(M) remain unaltered.

- **FOVR**: is set if B0 of N + 1(M) equals 1, otherwise it is left clear.
- **C**: is not used and will be left clear.
- **V**: will remain unchanged.

**Modification**  
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Machine Code Listing**

```
136 2 M N
```
LFP

Load Floating-point

**Function**

Load the floating-point number in \(N(M)\) and \(N + 1(M)\) to the floating-point accumulator A.

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>LFP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>(N(M))</td>
</tr>
</tbody>
</table>

\(N\) may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand \(N(M)\) refers to the first of two consecutive locations that hold
the floating-point number.

**Execution**

The LFP instruction loads the floating-point accumulator A with the double-length number in \(N(M)\) and \(N + 1(M)\).
The contents of \(N(M)\) and \(N + 1(M)\) are unaltered.

**FOVR** is set if \(B0\) of \(N + 1(M)\) equals 1, otherwise it is left clear.

**C** is not used and will be left clear.

**V** is not used and remains unchanged.

**Modification** This statement has an M-field. When modified, the least significant 15 bits of \(N + M\) are taken as
the operand. In the extended data mode, the least significant 22 bits of \(N + M\) are taken as the operand.

**Machine Code Listing**

136 0 M N

**Notes**

1. If operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower
data. (The symbol is given the value of the first location.)
**PLAN 1,2,3**

**Load Floating-point Zero**

**Function**
Clear the floating-point accumulator A and the exponent register.

**Format**
- *Operation Code*: LFPZ
- *Accumulator*: Blank
- *Operand*: Blank

**Execution**
The LFPZ instruction clears the floating-point accumulator A and the exponent register.

- *FOVR* is not used and will be left clear.
- *C* is not used and will be left clear.
- *V* is not used and remains unchanged.

**Modification**
This statement has no M-field.

**Machine Code Listing**

```
136 1 0 0
```
**MODE**

**Set Zero-suppression Mode**

**Function**
Set a mode number for zero-suppression in association with the CBD instruction.

**Format**

- **Operation Code**: MODE
- **Accumulator**: Blank
- **Operand**: N(M)

N may be:
- An absolute expression, e.g.
  - (a) A decimal or octal integer.
  - (b) A previously defined absolute symbol.

The operand N(M) specifies the mode number to be set and must have a value 0 or 1.

**Execution**

Before a binary number is converted to decimal characters by use of the CBD, the MODE function indicates whether or not zero-suppression is required.

The non-significant zeros are suppressed (i.e. converted to space characters) if N(M) = 1.

The non-significant zeros are not suppressed if N(M) = 0.

The mode should not be set to any value other than zero or one.

After converting the first non-zero character during the CBD instruction, the mode is reset to zero, so that any subsequent zeros will not be converted to spaces.

C is not used and will be left clear.

V is not used and remains unchanged.

**Modification**

This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Example**

This example shows the conversion and zero-suppression of a binary integer to six decimal characters.

```
| LABEL | OPERATION | ACC | 35 | 32 | 29 | 26 | 23 | 20 | 17 | 14 | 11 | 8 | 5 | 2 | 0 | OPERAND | 47 | 44 | 41 | 38 | 35 | 32 | 29 | 26 | 23 | 20 | 17 | 14 | 11 | 8 | 5 | 2 | 0 | MODE
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---
Notes
1 The initial setting of zero suppression mode when a program is loaded is indeterminate.
2 The setting of zero suppression mode resulting from a MODE instruction applies only to the program member issuing the instruction; other members are not affected.
3 The setting of zero suppression mode to zero by a CBD instruction on the conversion of the first non-zero character applies only to the program member issuing the instruction; other members are not affected.
4 If zero suppression mode was set to 0 when a CALL instruction came to be obeyed, it will remain set to 0. If zero suppression mode was set to 1 when a CALL instruction came to be obeyed, it will be left in an indeterminate state.
5 The setting of zero suppression mode at the time that a CALL instruction came to be obeyed is stored in the link accumulator. On returning from the subroutine by means of an EXIT instruction, zero suppression mode is left in the state it was in before the CALL, provided that:
   (a) the contents of the link accumulator when the EXIT instruction is obeyed are the same as when the subroutine was entered, and
   (b) the program member is operating in the same address mode and branch mode as when the subroutine was entered, and
   (c) if the program member is operating in compact mode and direct branch mode (15AM and DBM), negative modification of the EXIT instruction is not employed.

   In other circumstances, zero suppression mode is left in an indeterminate state by an EXIT instruction. (For a fuller discussion of this point, see the EXIT instruction.)
MOVE

PLAN 1,2,3

Copy N Words from the Address in X to the Address in X*

Function
Move a block of words from one position in store to another, leaving them in the same consecutive order.

Format

Operation Code
MOVE

Accumulator
X

Operand
N(M)

N may be:
An absolute expression in the range 0 to 4095, e.g.
(a) A decimal integer in the range 0 to 4095.
(b) An octal integer in the range #0 to #7777.
(c) A previously defined symbol with a value in the range 0 to 4095.

The operand N(M) specifies the number of words to be transferred, and must have a value in the range 0 to 511 (inclusive).

Execution
The MOVE instruction copies the word whose address is specified in the least significant 15 bits of X into the address specified in the least significant 15 bits of X*. Both addresses are then incremented by 1, and the procedure repeated until a total of N words have been transferred. In extended data mode, the least significant 22 bits of X and X* are taken as the addresses specified.

The original addresses in X and X* remain unchanged after execution, unless included in the area into which the copying is specified. A count of zero causes 512 words to be moved.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Examples

1 It is required to move a 75-word block starting at SERAL to a different area of store starting at INTER, both in lower data.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>01 02 03 04 05 06 07 08 09 10 11 12 13 14 15</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDN</td>
<td>6</td>
<td>SERAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDN</td>
<td>7</td>
<td>INTER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE</td>
<td>6</td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>

2 It is required to zeroize a 46-word area starting at REDOH, in lower data.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>01 02 03 04 05 06 07 08 09 10 11 12 13 14 15</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STOR</td>
<td></td>
<td>REDOH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDN</td>
<td>4</td>
<td>REDOH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDN</td>
<td>5</td>
<td>REDOH+1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE</td>
<td>4</td>
<td>46</td>
<td></td>
</tr>
</tbody>
</table>
Machine Code Listing
126  X M N

Notes
1 The two start addresses of the blocks must be loaded into consecutive accumulators before the instruction is obeyed. As the transfer is of a progressive type, care should be taken that the source and destination areas do not overlap (unless such overlap is specifically required).
Multiply and Accumulate

Function
Multiply the contents of X by the contents of N(M), add the signed contents of X* and put the double-length result in XX*.

Format

- **Operation Code**: MPA
- **Accumulator**: X
- **Operand**: N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to the location that contains the multiplier.

Execution

The MPA instruction causes the contents of X to be multiplied by the contents of N(M). The contents of X* are added to the less significant half of the result, and the double-length answer is left in XX*. The sign bit of X* is always left clear. The contents of N(M) are unaltered.

C is not used and will be left clear.

V will be set if both factors have B0 = 1, and all other bits = 0 (i.e. the fraction -1.0) and the initial content of X* is positive or zero.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

A positive length in yards, feet and inches is stored in locations YARDS, FEET and INS respectively. It is required to convert the whole quantity to inches and store the result in INS. Assume the answer does not exceed the capacity of one computer word (i.e. V will not be set). Locations CON12 and CON36 contain the conversion factors 12 and 36.
In this example, though the results of the two MPA instructions result technically in a double-length answer, due to the assumption that the total inches does not exceed the capacity of one computer word, the significant part of the result is contained entirely in X6.

**Machine Code Listing**

042 X M N

**Notes**

1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

2. The contents of X and of N(M) may be regarded as integers, fractions, or mixed numbers. Where fractions or mixed numbers are involved, the number of bits after the binary point in the product is equal to the sum of the numbers of bits after the binary points in the factors.
Multiply and Round

**Function**
Multiply the contents of the single-length word in X by the contents of N(M), leaving the rounded single-length answer in X.

**Format**
- **Operation Code**: MPR
- **Accumulator**: X
- **Operand**: N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.
3. A literal (not PLAN 1).

The operand N(M) refers to the location that contains the multiplier.

**Execution**
The MPR instruction multiplies the contents of X by the contents of N(M), producing a double-length product in XX*. One is then added to the B1 position of X*, causing a carry into X if B1 of X* is one. The result is thus a single-length product in X. The sign bit of X* is always made a zero.

C is not used and will be left clear.

V will be set if both factors have B0 = 1 and all other bits = 0 (i.e. the fraction -1.0). In this case, the result will have -1.0 in X.

**Modification**
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Example**
Accumulator X3 contains 34 and location COLET contains 4/37 as a fraction. It is required to calculate the product of 34 and 4/37 to the nearest whole number, and to store the result in ANSER.

**Machine Code Listing**
041 X M N

**Notes**
1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
2. The contents of X and of N(M) may be regarded as integers, fractions, or mixed numbers. Where fractions or mixed numbers are involved, the number of bits after the binary point in the (double-length) product is equal to the sum of the numbers of bits after the binary points in the factors.
Multiply

Function Multiply the contents of X by the contents of N(M) and place the double-length product in XX*.

Format

Operation Code MPY
Accumulator X
Operand N(M)

N may be:
1  A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
       decimal or octal integer.
2  An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range
       0 to 4095.
3  A literal (not PLAN 1),
   The operand N(M) refers to the location that contains the multiplier.

Execution

The MPY instruction multiplies the contents of X by the contents of N(M). The double-length product is
left in XX*. The sign bit of X* is always set to 0.

C is not used and will be left clear.

V will be set if both factors have B0 = 1 and all other bits = 0 (i.e. the fraction -1.0). In this case, the
result in X, will have B0 = 1, B1 to B23 = 0, and in X*, B0 to B23 = 0 (i.e. the double-length fraction
-1.0).

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are
taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the
operand.

Machine Code Listing

040 X M N

Notes

1 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the
symbol (not PLAN 1).
2 The contents of X and of N(M) may be regarded as integers, fractions, or mixed numbers. Where
fractions or mixed numbers are involved, the number of bits after the binary point in the product
is equal to the sum of the numbers of bits after the binary points in the factors.
PLAN 3.4

Move N Characters from the Character-address specified in X to the Character-address specified in X*. (Available on 1902A, 1903A and Processors with Extended Data Mode Facilities)

Function
Move a string of characters from one position in store to another, leaving them in the same consecutive order.

Format
Operation Code: MVCH
Accumulator: X
Operand: N(M)

N may be:
An absolute expression in the range 0 to 4095, e.g.
(a) A decimal integer in the range 0 to 4095.
(b) An octal integer in the range #0 to #7777.
(c) A previously defined symbol with a value in the range 0 to 4095.
The operand N(M) specifies the number of characters to be moved, and should not exceed 511.

Execution
The MVCH instruction copies the character whose address is specified in X into the character position whose address is specified in X*. Both character addresses are then incremented by one and the procedure is repeated until a total of N(M) characters have been transferred. After execution, the addresses in X and X* will be left pointing to the character following the last character transferred. A count of zero will cause 512 characters to be moved. When operating in compact mode, the least significant 15 bits of X and X* are taken as specifying the word addresses and the contents of B2 to B8 are preserved; when in extended data mode, the least significant 22 bits are taken. In either case, B0 and B1 are taken as specifying the character position within the word.

C is not used and will be left clear.
V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example
It is required to move a 136-character block starting at KAYB + 1.2 in upper data to a different area of store starting at REDOH.0 in lower data.

| LABEL | OPERATION | ACC. | 12345 | 67 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 48 | 52 | 56 | 60 | 64 | 68 | 72 |
|-------|-----------|------|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| LOX   | 6         | 'KAYB+1' | '1.2'  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| LDX   | 7         |       | 'REDOH' |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| MVCH  | 6         |       | 1.36   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Machine Code Listing
116 X M N

Notes
1 The two start addresses of the character blocks must be loaded into consecutive accumulators before the instruction is obeyed. As the transfer is of a progressive type, care should be taken that the source and destination areas do not overlap (unless such an overlap is specifically required). In particular, where X and X* are included in the source or destination area, the result is undefined.
NEFPS

Negate Extended Floating-point Accumulator and Store

Function
Negate the extended floating-point accumulator and store the normalised result in N(M) to N + 3(M).

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>NEFPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of four consecutive locations that hold the extended floating-point number.

Execution

The NEFPS instruction will generate two machine code instructions. The first instruction generated negates the extended floating-point accumulator, then stores the contents of the standard floating-point accumulator in N(M) to N + 1(M). The second instruction stores the contents of the accumulator extension in N + 2(M) to N + 3(M).

Fovr if floating-point overflow occurs at any time during the execution of the instruction Fovr is set, otherwise it will remain unchanged.

C is not used and will be left clear.

V will be set if floating-point overflow occurs at any time during the execution of the instruction or Fovr is set when the instruction has been executed otherwise it will remain unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing

137 4 M N
137 2 M N

Notes
NEFPS N(M) is equivalent to
NFPS N(M)
SAE N + 2(M)
NFPS

PLAN 4  (#XPLT ONLY)  Negate Floating-point Accumulator and Store
Function  Negate the extended floating-point accumulator and store the normalised result in N(M) to N + 1(M).

Format  

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>NFPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of two consecutive locations in which the contents of the standard floating-point accumulator are to be stored.

Execution

The NFPS instruction negates the extended floating-point accumulator. The contents of the standard floating-point accumulator are stored in N(M) to N + 1(M).

**FOVR**  if floating-point overflow occurs at any time during the execution of the instruction **FOVR** is set otherwise it will remain unchanged.

C  is not used and will be left clear.

V  will be set if floating-point overflow occurs at any time during the execution of the instruction or **FOVR** is set when the instruction has been executed otherwise it will remain unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing

137  4  M  N
Load the Complement of N

Function
Load X with the complemented value of N(M).

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Accumulator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>NGN</td>
<td>X</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the value which is to be complemented and loaded into X.

Execution
The NGN instruction loads the complement of the value of N(M) into X. If C was left set by the previous instruction, I is subtracted from the result in X. The contents of N(M) are unaltered.

If the operand is the symbolic name of a location previously defined under a #LOWER directive, the address of that location will be complemented and loaded into X. If the operand is written as a symbolic identifier previously met by #DEFINE or #SET directives, the value that was assigned to that identifier is complemented and loaded into X.

C is used and will be left clear.

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing
102 X M N

Notes
1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
Load the Complement of N, with Carry

Function
Load into accumulator X the complemented value of N(M), setting C if carry occurs.

Format
   Operation Code  NGNC
   Accumulator     X
   Operand         N(M)

N may be:
1) A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2) An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the value which is to be complemented and loaded into X.

Execution
Effectively the NGNC instruction loads the complement of the value of N(M). If C was left set by the previous instruction, 1 is subtracted from the result in X. At the conclusion of this instruction, B0 of the result will always be zero.

If the operand is the symbolic name of a location previously defined under a #LOWER directive, then the address of that location will be complemented and loaded into X. If the operand is written as a symbolic identifier previously set by #DEFINE or #SET directives, the value that was assigned to that identifier is complemented and loaded into X.

C is used and will be set if carry occurs (see Note 2). See Chapter 1, page 6.
V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing
106  X  M  N

Notes
1) If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
2) C will always be set except when N(M) is 0 and C was not set initially.
NGS

Negative Load into Store

Function
Store the complement of the contents of X in N(M).
Store the complement of the contents of XX* in N(M) and N + 1(M).

Format

<table>
<thead>
<tr>
<th>Operating Code</th>
<th>NGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X or XX*</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

If X is specified, N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

If X is specified, the operand N(M) refers to the location in which the complement of the contents of X are to be stored.

If XX* is specified, N may be types 1 and 2 above. When N is type 2 it should be in the range 0 to 4094. The operand N(M) refers to the first of two consecutive locations in which the complement of the contents of XX* are to be stored.

Execution

If X is specified, the NGS instruction stores the complement of the contents of X in N(M). If C was left set by the previous instruction, 1 is subtracted from the result in N(M). The contents of X are unaltered.

If XX* is specified, the NGS instruction stores the complement of the contents of XX* in N(M) and N + 1(M). If C was left set by the previous instruction, 1 is subtracted from the result in N(M) and N + 1(M). The contents of XX* are unaltered. B0 in X* must be 0, otherwise an incorrect result will be produced.

C is used and will be left clear.

V will be set if X contains #40000000 and C was not set when the NGS instruction came to be obeyed.

The contents of X are unaltered.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing

When X is specified: 012 X M N
When XX* is specified: 016 X* M N + 1
012 X M N
Notes

1. If X is specified and operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

If XX* is specified and operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data (the symbol is given the value of the first location). Not PLAN 1.

2. NGS XX* N(M)
is equivalent to
NGSC X* N + 1(M)
NGS X N(M)
NGSC

Negative Load into Store with Carry

Function      Store the complement of the contents of X in N(M), setting C if carry occurs.

Format        
   Operation Code   NGSC
   Accumulator      X
   Operand          N(M)

   N may be:
   1. A relative expression which refers to a lower data location, e.g.
      (a) A symbolic name referring to a lower data location.
      (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.
   2. An absolute expression in the range 0 to 4095, e.g.
      (a) A decimal integer in the range 0 to 4095.
      (b) An octal integer in the range #0 to #7777.
      (c) A previously defined absolute symbol in the range 0 to 4095.

   The operand N(M) refers to the location in which the complement of the
   contents of X are to be stored.

Execution

The complement of the contents of X are loaded into N(M). If C was left set by the previous instruction,
1 is subtracted from the result in N(M). The contents of X are unaltered. At the conclusion of this
instruction B0 of the result will always be zero.

C    is used and will be set if carry occurs. See Chapter 1, page 6.

V    is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are
taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the
operand.

Example

In the following example, the complement of a triple-length word in X5, X6 and X7 is to be loaded into
BOB, BOB + 1 and BOB + 2.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NGSC</td>
<td>7</td>
<td>BOB+2</td>
</tr>
<tr>
<td></td>
<td>NGS.</td>
<td>56</td>
<td>BOB</td>
</tr>
</tbody>
</table>

Machine Code Listing

016 X M N
Notes

1 When dealing with double-length numbers in PLAN 3 it is permissible to use the double-length version of the NGS instruction, but when working with numbers of a greater size the NGSC instruction must be used for all but the most significant two words of the operands.

2 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
Negative Load into X

Function
Load the complement of the contents of N(M) into X.
Load the complement of the contents of N(M) and N + 1(M) into XX*.

Format

- **Operation Code**: NGX
- **Accumulator**: X or XX*
- **Operand**: N(M)

If X is specified, N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to a location the complement of whose contents is to be loaded into X.

If XX* is specified, N may be types 1 and 2 above. When N is type 2 it should be in the range 0 to 4094. The operand N(M) refers to the first of two consecutive locations the complement of whose contents is to be loaded into XX*.

Execution

If X is specified, the complement of the contents of N(M) is placed in X. If C was left set by the previous instruction, 1 is subtracted from the result in X. The contents of N(M) are unaltered.

If XX* is specified, the complement of the contents of N(M) and N + 1(M) is loaded into XX*. If C was left set by the previous instruction, 1 is subtracted from the result in XX*.

C is used and will be left clear. The contents of N(M) and N + 1(M) are unaltered.

V will be set if N(M) contains #40000000 and C was not set initially.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing

When X is specified: 002 X M N
When XX* is specified: 006 X* M N + 1
002 X M N
Notes

1 If X is specified and operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

   If XX* is specified and operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data (the symbol is given the value of the first location). Not PLAN 1.

2 NGX  XX*  N(M)

   is equivalent to

   NGXC  X*  N + 1(M)

   NGX  X  N(M)
NGXC

Negative Load into X with Carry

Function
Load the complement of N(M) into X, setting C if carry occurs.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>NGXC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol with a value in the range 0 to 4095.
3. A literal (not PLAN 1).

The operand N(M) refers to the location the complement of which is to be loaded into X.

Execution

The complement of the contents of N(M) are loaded into X. If C was left set by the previous instruction, 1 is subtracted from the result in X.

The contents of N(M) are unaltered. At the conclusion of this instruction, B0 in the result will always be zero.

C is used and will be set if carry occurs. See Chapter 1, page 6.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

In the following example, the complement of a triple-length number held in DER, DER + 1 and DER + 2 is to be loaded into X4, X5 and X6.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>M</th>
<th>20</th>
<th>16</th>
<th>24</th>
<th>8</th>
<th>6</th>
<th>4</th>
<th>8</th>
<th>24</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NGXC</td>
<td>6</td>
<td>DER + 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NGX</td>
<td>8</td>
<td>DER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Machine Code Listing

006  X  M  N

160  Chapter 4
Notes

1 When dealing with double-length numbers in PLAN 3, it is possible to use the double-length version of the NGX instruction, but when working with numbers of greater size, the NGXC instruction must be used for all but the most significant two words of the operands.

2 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
Normalize

Function Normalize a single- or double-length floating-point number.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>NORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X or XX*</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
An absolute expression in the range 0 to 4095, e.g.
(a) A decimal integer in the range 0 to 4095.
(b) An octal integer in the range #0 to #7777.
(c) A previously defined symbol with a value in the range 0 to 4095.
The operand N(M) specifies the value of the exponent plus 256.

Execution

Where X is specified, the NORM instruction normalizes a floating-point number whose single-length argument is held in X, and N(M) specifies the value of the exponent plus 256. The result is left as a floating-point number, with single-length argument, in X and X*.

The effects of the instruction may be divided into the following steps:

1. It clears X*.
2. If V is clear, the contents of X are shifted arithmetically one place left and at the same time N(M) is reduced by 1. This process is repeated until either
   (a) the next shift would change the sign bit of X, or
   (b) N(M) is counted down to zero, or
   (c) the argument of the number is shown to have been zero from the start.

When condition (a) is satisfied, the argument is in normalized form and the final value of N(M) is deposited in the last nine bits of X*. If condition (b) or (c) occurs then the result in X and X* is floating-point zero with both argument and exponent cleared.

If V is set originally, then the contents of X are given a special unrounded shift right arithmetic for one-place and N(M) is increased by 1. The final value of N(M) is deposited into the last nine bits of X*.

If the final value of N(M) is greater than 511, then the Overflow register V is set.

Where XX* is specified, the NORM instruction normalizes a floating-point number whose double-length argument is held in X and X* and whose exponent is the value N(M). The action and use of this instruction are similar to the single-length normalize instruction except that X* is not cleared at the start and the shifts are double-length. The result after shifting is rounded to 38 bits and the exponent is deposited in B15 to B23 of X*.

C is not used and will be left clear.

V is used, see above.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.
Examples

1. It is required to normalize the floating-point number whose single-length argument is in NUM and whose characteristic is in NUM + 1.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>45</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX</td>
<td>7</td>
<td>NUM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX</td>
<td>2</td>
<td>NUM+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORM</td>
<td>7</td>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(PLAN73ONLY)</td>
</tr>
<tr>
<td>STO</td>
<td>70</td>
<td>NUM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first two instructions load the argument into X7 and the exponent into X2 respectively. The third instruction brings the argument in X7 within the range

\[ 1 > \text{argument} \geq \frac{1}{2}, \text{or} \]
\[ -\frac{1}{2} > \text{argument} \geq -1 \]

and adjusts the exponent in X2 accordingly.

The normalized floating-point number is now contained in accumulators X7 and X0. The fourth instruction stores this normalized floating-point number in NUM and NUM + 1.

The NORM instruction is usually written as above, and is equivalent to NORM 7 0(2).

2. It is required to normalize the double-length floating-point number in locations RED and RED + 1.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX</td>
<td>67</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(PLAN73ONLY)</td>
</tr>
<tr>
<td>ANDX</td>
<td>7</td>
<td>'#77777.0000'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX</td>
<td>2</td>
<td>RED+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORM</td>
<td>67</td>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(PLAN73ONLY)</td>
</tr>
<tr>
<td>STO</td>
<td>67</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first instruction loads the double-length floating-point number into X6 and X7. The second instruction clears bits 15 to 23 of X7 (i.e. the exponent part of the number) thus leaving the argument isolated in X6 and X7. The third instruction loads the exponent into X2. The fourth instruction brings the argument in X6 and X7 within the range

\[ 1 > \text{argument} \geq \frac{1}{4}, \text{or} \]
\[ -\frac{1}{4} > \text{argument} \geq 1 \]

and adjusts the exponent accordingly.

The normalized floating-point number is now contained in X6 and X7 with B15 and B23 of X7 holding the exponent. The fifth instruction stores this normalized floating-point number in RED and RED + 1.

The NORM instruction is usually written as above, and is equivalent to NORM 67 0(2).

3. A double-length, fixed-point fraction is in locations FRAC and FRAC + 1. Convert it to a standard floating-point number in locations TINY and TINY + 1.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX</td>
<td>67</td>
<td>FRAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(PLAN73ONLY)</td>
</tr>
<tr>
<td>NORM</td>
<td>67</td>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STO</td>
<td>67</td>
<td>TINY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(PLAN73ONLY)</td>
</tr>
</tbody>
</table>

The first instruction loads the fraction into X6, 7. Then the NORM shifts it until it lies in the range for a normalized floating-point argument, subtracting 1 from 256 for each place of shift and thus creating the correct characteristic; the argument is then rounded to 38 places and the characteristic set into bits 15 to 23 of X7.

Finally, the floating-point number is copied into store.

Machine Code Listing
If X is specified: 114 X M N
If XX* is specified: 115 X M N
**NULL**

No Operation

**Function** This instruction has no effect.

**Format**
- *Operation Code*: NULL
- *Accumulator*: Blank
- *Operand*: Blank

**Execution**
None.

*C* is not used and remains unchanged.

*V* is not used and remains unchanged.

**Modification** The NULL statement has an M-field.

**Example**

| LABEL | OPERATION | ACC | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | NULL      |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Machine Code Listing**

123 0 0 0

**Notes**

1. This may be used as a dummy instruction. For example, the NULL instruction used by the PLAN compiler; when a statement is encountered with an unrecognized operation code, a NULL instruction is inserted and compilation continues.
OBEY

Obey the Instruction at N

Function
The instruction whose location is specified by the operand is obeyed as if it occupied the position of the OBEY instruction.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>OBEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to the location which contains the instruction to be obeyed.

Execution
The instruction, whose address is specified in N(M), is obeyed as if it occupied the position of the OBEY instruction. A return is then made to the instruction succeeding OBEY, unless the specified instruction was a branch, itself transferring control to some other point.

C is not used and remains unchanged.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Examples

1

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>X</th>
<th>N</th>
<th>M</th>
<th>V</th>
<th>U</th>
<th>W</th>
<th>R</th>
<th>G</th>
<th>K</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDN</td>
<td>1</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBEY</td>
<td>1</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STO</td>
<td>3</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This sequence causes the instruction in COND + 16 to be obeyed, after which the instruction STO 3 RES is carried out (unless, of course, COND + 16 contains a branch instruction transferring control to another part of the program).
In this example the instruction held in CONST modified by the contents of X3 is obeyed (e.g. if X3 contains 2, the instruction obeyed would be held in CONST + 2, that is CALL 1 GROSS, and the EXIT would be to the information following the OBEY).

Machine Code Listing
023 0 M N

Notes
1 If the obeyed instruction is a branch instruction with a relative operand (i.e. of the form \( \text{LOC} + n \)), the location represented by \( \text{LOC} \) will be that of the obeyed instruction and not that occupied by the OBEY instruction.

2 In extended branch mode, if the obeyed instruction is a relative branch, the destination address of the relative branch is determined relative to the location holding the OBEY instruction, not relative to the location holding the branch instruction.

3 If the obeyed instruction is a CALL, then the link address stored in the specified accumulator for the CALL, is that of the instruction following OBEY.

4 If the obeyed instruction is another OBEY, then the second OBEY will operate as if it were the first encountered and has been in the first's location, i.e. any return will be to the first OBEY.

5 If the obeyed instruction is a SUSMA, then the instruction performed as a result of the SUSMA will be that located next or next but one after the OBEY, not that next or next but one after the SUSMA.

6 The operand may appear under #LOWER. By allowing the programmer to OBEY instructions held in a data area, it also, implicitly, permits alteration of a program instruction by those instructions normally affecting that area (e.g. arithmetic and logical instructions).

7 Though the OBEY instruction does not consider or alter the state of V or C, the obeyed instruction may do so.

8 An OBEY instruction will almost always be modified as its most common use will be as a program switch.

9 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
OFF

Unset Switch

**PLAN 3**
*(Macro-instruction)*

**Function**  
Set bit N of word 30 (the switch word) equal to zero.

**Format**

- **Operation Code**: OFF
- **Accumulator**: X
- **Operand**: N(M)

N may be:
- An absolute expression in the range 0 to 1023, e.g.
- A decimal integer in the range 0 to 1023.
- An octal integer in the range #0 to #1777.
- A previously defined symbol with a value in the range 0 to 1023.

The operand N(M) specifies the bit position which is to be made equal to zero, and must have a value in the range 0 to 23.

**Execution**

OFF is a macro-instruction which results in the generation of three machine-code instructions. Its effect is to set a specified bit (0 to 23) of word 30 equal to 0, leaving all the other bits unchanged.

The original contents of X are destroyed by the macro-instruction which should not be used if C was left set by the previous instruction obeyed.

C is used and will be left clear (see above).

V is not used and remains unchanged.

**Modification**  
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Example**

It is required to unset bit 13 of the switch word (word 30).

| LABEL | OPERATION | ACC  | 12 | 33 | 16 | 20 | 24 | 28 | 32 | 16 | 40 | 44 | 48 | 52 | 56 | 60 | 64 | 68 | 72 |
|-------|-----------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | OFF       | 6    | 13 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

The instruction will result in the generation of the following series of machine-code instructions:

| LABEL | OPERATION | ACC  | 16 | 20 | 24 | 28 | 32 | 16 | 40 | 44 | 48 | 52 | 56 | 60 | 64 | 68 | 72 | 76 | 80 |
|-------|-----------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | NGNC      | 6    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|       | SRC       | 6    | 13 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|       | ANDS      | 6    | 30 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

The original contents of X6 are destroyed.

**Machine Code Listing**

106 X 0 1
112 X M N
030 X 0 30
ON

Set Switch

Function
Set bit N of word 30 (the switch word) equal to 1.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Accumulator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>X</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
- An absolute expression in the range 0 to 1023, e.g.
  - A decimal integer in the range 0 to 1023.
  - An octal integer in the range #0 to #1777.
  - A previously defined symbol with a value in the range 0 to 1023.

The operand N(M) specifies the bit position which is to be made equal to one, and must have a value in the range 0 to 23.

Execution

ON is a macro-instruction which results in the generation of three machine-code instructions. Its effect is to set a specified bit (0 to 23) of word 30 equal to 1, leaving all the other bits unchanged.

The contents of X are destroyed by the macro-instruction.

C is not used and will be left clear (see above).

V is not used and remains unchanged.

Modification

This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

It is required to set bit 17 of the switch word (word 30) equal to 1.

| LABEL | OPERATION | ACC   | N  | M  | I  | J  | K  | L  | OPERAND | B  | C  | D  | E  | F  | G  | H  | I  | J  | K  | L  | 1  | 2  | 3  | 4  |
|-------|-----------|-------|----|----|---|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ON    | 5         | 1,7   |    |    |   |    |    |    |         |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

The instruction will result in the generation of the following series of machine-code instructions:

| LABEL | OPERATION | ACC   | N  | M  | I  | J  | K  | L  | OPERAND | B  | C  | D  | E  | F  | G  | H  | I  | J  | K  | L  | 1  | 2  | 3  | 4  |
|-------|-----------|-------|----|----|---|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| LDCX  | 5         | M5001 |    |    |   |    |    |    |         |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| SRL   | 5         | 1,7   |    |    |   |    |    |    |         |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| ORS   | 5         | 30    |    |    |   |    |    |    |         |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

The original contents of X5 are destroyed.

Machine Code Listing

124 X 0 256
112 X M N + 1024
031 X 0 30

168 Chapter 4
PLAN 1,2,3

Logical INCLUSIVE OR Direct Operand to X

**Function** Perform a 'logical INCLUSIVE OR' operation on the contents of X using the value of N(M).

**Format**

- **Operation Code**: ORN
- **Accumulator**: X
- **Operand**: N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specified the value to be used in the 'logical INCLUSIVE OR' operation.

**Execution**

A 'logical INCLUSIVE OR' operation is performed with the contents of X and the value of N(M). Both words are regarded as 24-bit patterns rather than as numeric quantities. The instruction places a 1-bit in X in positions where either X or N(M), or both, have a 1-bit. The contents of N(M) are unaltered.

If the operand is the symbolic name of a location previously defined under a #LOWER directive, the address of that location will be used as the appropriate bit pattern. If the operand is written as a symbolic identifier previously set by #DEFINE or #SET directives, the value that was assigned to that identifier is used as the appropriate bit pattern.

C is not used and will be left clear.

V is not used and remains unchanged.

**Modification** This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Examples**

1. It is required to set bits 12 and 13 of X6 equal to 1.

   Original contents of X6: 000 000 101 111 000 000 000

   Operation: ORN 0 #000

   Result in X6: 000 000 101 111 110 000 000 000

   Chapter 4
This instruction results in bits 12 to 14 and bits 18 to 20 of X3 being set to 1, the other bits in X3 remaining unchanged.

3 Accumulator 2 contains an index word. It is required to alter the modifier in it to the address of SYMBOL which is in lower data, leaving the counter unchanged.

| LABEL | OPERATION | ACC.  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------|-----------|-------|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| 3     | $BRN$     |       |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 3     | $7070$    |       |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |

| LABEL | OPERATION | ACC.  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------|-----------|-------|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| 2     | $ANDX$    |       |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 2     | $7777,00000$ |     |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 2     | $BRN$     |       |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| 2     | SYMBOL    |       |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |

Machine Code Listing

121 X M N

Notes

1 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
PLAN 1, 2, 3

Function
Perform a 'logical INCLUSIVE OR' operation on the contents of N(M) and the contents of X, leaving the result in N(M).

Format
Operation Code
ORS
Accumulator
X
Operand
N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range 0700 to 07777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the location the contents of which are to be used in the 'logical INCLUSIVE OR' operation.

Execution
A 'logical INCLUSIVE OR' operation is performed with the contents of N(M) and the contents of X. Both words are regarded as 24-bit patterns rather than numeric quantities. The instruction places a 1-bit in N(M) where either X or N(M), or both, have a 1-bit. The contents of X are unaltered.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Examples
1. Original contents of HAW: 110 010 001 000 111 101 010 110
   Original contents of X2: 001 111 011 011 100 000 111 100
   
   Operation:
   
   Result in HAW: 111 111 011 011 111 101 111 110
   (X2 is unchanged.)

2. Location INDEX contains an index word with a count of zero. It is required to make the count equal to 27.

Chapter 4
3. Make the sign bit in location SAGE equal to 1.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>ASCII</th>
<th>D1</th>
<th>D0</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D6</th>
<th>D7</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDCT</td>
<td>0</td>
<td>#ACC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ORS</td>
<td>0</td>
<td>SAGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Machine Code Listing**

031 X M N

**Notes**

1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
PLAN 1,2,3

Logical INCLUSIVE OR into X

Function
Perform a 'logical INCLUSIVE OR' operation on the contents of X and the contents of N(M), leaving the result in X.

Format
Operation Code  ORX
Accumulator  X
Operand  N(M)

N may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.
2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.
3 A literal (not PLAN 1).

The operand N(M) refers to the location the contents of which are to be used in the 'logical INCLUSIVE OR' operation.

Execution
A 'logical INCLUSIVE OR' operation is performed with the contents of X and the contents of N(M). Both words are regarded as 24-bit patterns rather than numeric quantities. The instruction places a 1-bit in X where either X or N(M), or both, have a 1-bit. The contents of N(M) are unaltered.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N(M) are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example
Original contents of X2: 110 001 010 000 110 010 110 100
Original contents of MASK: 001 100 000 101 101 111 001 011

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ORX</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation: [ORX 12  MASK]
```

Result in X2: 111 101 010 101 111 111 111 111

(MASK is unchanged.)

Machine Code Listing
021 X M N

Notes
1 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
Invert Switch and Indicate

**Function**  Invert the state of bit \( n \) of word 30 (the switch word).

**Format**

- **Operation Code**: OVER
- **Accumulator**: X
- **Operand**: N(M)

\( N \) may be:

An absolute expression in the range 0 to 1023, e.g.

(a) A decimal integer in the range 0 to 1023.
(b) An octal integer in the range \#0 to \#1777.
(c) A previously defined symbol with a value in the range 0 to 1023.

The operand N(M) specifies the bit position to be inverted, and must have a value in the range 0 to 23.

**Execution**

OVER is a macro-instruction which results in the generation of four machine-code instructions. Its effect is to invert the state of bit \( n \) of word 30 (the switch word) and to indicate the state of the inverted bit in the specified accumulator. If the inverted bit is a 1-bit, then the content of X will be non-zero. If the bit is now zero then the content of X will be equal to zero.

The original contents of X are destroyed by the macro-instruction.

C is not used and will be left clear.

V is not used and remains unchanged.

**Modification** This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Example**

It is required to test the state of bit 10 of the switch word (word 30) and to invert the state of that bit. If the switch is set, branch to SET.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>PROD</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER</td>
<td>S. 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@R8</td>
<td>5, UNSET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The OVER macro-instruction above generates the following series of machine-code instructions:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>PROD</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDCT</td>
<td>S. #120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRL</td>
<td>S. 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERS</td>
<td>S. 30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANDK</td>
<td>S. 30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#
The ANDX instruction in this example will leave a 1-bit in B10 of X5 if the inverted state of B10 in word 30 is 1 (i.e. if B10 of word 30 was originally zero, there will be a 1-bit in B10 of X5).

It is important to remember this fact when testing the original state of the switch.

**Machine Code Listing**

124 X 0 #400
112 X M N + 1024
032 X 0 30
020 X 0 30

**Notes**

1 The state of the inverted bit can be found by testing the content of the specified accumulator (e.g. with a BNZ instruction).
Initiate Peripheral Action

Function
Initiate action on a specified peripheral.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>PERI</th>
</tr>
</thead>
</table>

Accumulator
This may be:

1. The peripheral unit number.
2. The address of an accumulator which contains the unit number.

Operand
N(M)
N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) refers to a location which is the first word of the control area which contains the information necessary for Executive to perform the peripheral action.

Execution
The PERI instruction causes a program entry to Executive. Executive, dependent on the contents of the control area, will initiate the peripheral operation after checking that the specified peripheral is free.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERI</td>
<td>3. PT. CON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERI</td>
<td>1. ERGE N (2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The above example shows the format of the PERI instruction. Note that the location specified in the operand field is the first word of the control area. In the example above, if B0 of the first word were 0 then the unit number in the first case would be 3 and in the second case 1 (i.e. equivalent to the address of the specified accumulator). If B0 were 1, the unit number would be equivalent in the first case to the least significant six bits of accumulator 3, and in the second case to the least significant six bits of accumulator 1.
Machine Code Listing

157 X M N

Notes

1 (a) If bit 0 of the first word of the control area is 0, then the accumulator field specifies the peripheral unit number (with direct access devices, the file number).

(b) If bit 0 of the first word of the control area is 1, then the accumulator field specifies the address of an accumulator which contains in its least significant six bits the peripheral unit number (with direct access devices, the file number). Except as specified in (c), bits 0 to 17 of the specified accumulator must be set to zero.

(c) In this latter case, if the PERI instruction is being used in relation to magnetic tape in open tape modes #400 or #600 (i.e., calling for a scratch tape; see Appendix 4), then the accumulator may also specify the properties required of the tape. It will then have the following format:

| Bits 0 to 8 | Set to zero. |
| Bit 9      | Set to zero if the tape to be opened is required to have at least all the properties specified in bits 10 to 17. Set to one if the tape to be opened is to have none of the properties specified in bits 10 to 17. |
| Bits 10 to 17 | Contain the property code defining the tape to be opened. |
| Bits 18 to 23 | The program’s unit number of the tape to be opened. |

If bits 9 to 17 of the accumulator are set to zero, then no properties are specified for the tape, and Executive will open the first available scratch tape.

In other modes, including open tape modes which call for a named tape, bits 0 to 17 of the accumulator must be set to zero, otherwise the effect is undefined.

This facility applies to environments with dual program and multi-program Executives. If the program is run with an Executive on which the properties system has not been implemented, bits 9 to 17 of the accumulator are ignored by Executive.

Details of the property codes for magnetic tape systems may be found in the 1900 Series reference manual Magnetic Tape, Chapter 7.

(d) If the PERI instruction refers to a multiplexor, bit 0 of the first word of the control area is set to 1, and the contents of the specified accumulator will be as follows:

| Bits 0 and 1 | Set to zero. |
| Bits 2 to 17 | Line number (where applicable). |
| Bits 18 to 23 | The program’s unit number of the multiplexor. |

2 The form and content of the control area for the various peripherals will be found in the following publications:

Paper Tape Reader
Paper Tape Punch
Line Printer
Card Reader
Card Punch
Magnetic Tape
Cassette Tape
Exchangeable Disc Store
Magnetic Card File
Fixed Disc Store

Basic Peripherals manual
Magnetic Tape manual
Cassette Tape manual
Direct Access manual

Chapter 4
Interrogating Typewriter Facility
Multiplexor
Digital Incremental Plotter

Data Communications and Interrogation manual

Graph Plotter manual.

The control areas for basic peripherals, magnetic storage devices and the digital incremental plotter are summarized in Part 4, Appendix 4.

3 When a PERI instruction is used to initiate action on an interrogating typewriter or a multiplexor, a 'direct response' is given in word 9 of the program. This 'direct response' should be examined before the program member issues another PERI or ALLOT instruction, preferably immediately after the PERI is issued. The format of the 'direct response' in word 9 is as follows:

Bit 0    Set to 0 if Executive has rejected and ignored the order.

          Set to 1 if Executive has accepted the order.

Bits 1 to 23  Undefined.

Possible reasons for the rejection of the PERI instruction with these devices are listed in the Data Communications and Interrogation manual.

4 Executive keeps a queue of read and write instructions to direct access devices. When the queue is full, a program member giving a PERI instruction to initiate reading from or writing to a direct access device is suspended until the instruction can be accommodated in the queue. PERI instructions specifying a particular file number are obeyed from the queue in the sequence in which they were issued by the program, but this must not be assumed for instructions specifying different file numbers, even if these are in fact associated with the same file. If two PERI instructions specifying the same file number are given by a program, the program member which issued the second is suspended until the first of the instructions is completed.

5 With peripherals other than direct access devices, if two PERI instructions are given specifying the same peripheral unit, the program member which issues the second is suspended until the first of the instructions is completed.
PLAN 1,2,3,4

Function
Release the specified peripheral from the program, and make it available for re-allocation.

Format

**Operation Code**
REL

The rest of the instruction can take three possible forms:

1 **Accumulator**
   Blank

2 **Operand**
The symbolic name of the relevant peripheral (LP0, TP2, etc.).

3 **Accumulator**
The program's unit number of the relevant peripheral.

   N(M), where N(M) is the type number of the peripheral:

   0 = Paper Tape Reader
   1 = Paper Tape Punch
   2 = Line Printer
   3 = Card Reader
   4 = Card Punch
   20 = Digital Incremental Plotter.

4 **Operand**
   X, where X is an accumulator whose least significant six bits represent the unit number of the relevant peripheral.

   N(M), where N(M) = 256 + the type number of the peripheral as specified in 2.

PLAN 1 Compilers will not accept operands of type 1.

Execution
The REL instruction causes the unit specified to be released from the program and made available for allocating to that or any other program. The unit may also be disengaged. This instruction cannot be used for a magnetic tape unit or file storage devices.

Any transfer of information which is in progress is completed before the unit is released and the program is suspended pending completion. On completion of the transfer, Executive releases the peripheral and records the release on the console typewriter; the program then continues automatically.

- **C** is not used and will be left clear.
- **V** is not used and remains unaffected.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In extended data mode, the least significant 22 bits of N + M are taken as the operand. See Note 1 below.

Examples

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>TP2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REL</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REL</td>
<td>5 257</td>
<td></td>
<td>IXS CONTAINS TP2</td>
</tr>
</tbody>
</table>

In each case a Paper Tape Punch, TP2, is released from the program.

Machine Code Listing
151 X M N
Notes

1 The operand is modifiable, except in the case of a type 1 operand. If the operand is of types 2 or 3, the unit type may be modified. Note that a type 2 operand would be interpreted as a type 3 operand if B15 was set in the modifier.

2 If a type 3 operand is used, bits 0 to 17 of the accumulator specified by X must be set to zero.

3 This instruction is intended primarily for use on dual and multiprogram processors. It confers no advantages when used on a single-program processor.

4 If an attempt is made to operate a peripheral unit after it has been released the instruction will be treated as illegal.
PLAN 3.4
(Magnetic Tape Macro-instruction)

Function Rewind a magnetic tape.

Format
- Operation Code: REW
- Accumulator: A decimal digit in the range 0 to 7 specifying the program's unit number for the magnetic tape unit.
- Operand: Blank

Execution
REW rewinds the tape on the specified unit to a position immediately preceding the header label.

Modification The REW macro-instruction has no M-field.

Machine Code Listing
157 X 0 n LT
when n is the relative address in the literals table of the first of two consecutive locations which hold a control area of the form:
- first word: 5/ #7
- second word: 0

Notes
1 This is one of a set of magnetic tape macro-instructions which permit the execution of simple basic functions on magnetic tape without requiring the programmer to set up control areas. (See also BSP, BTM, CLOSE, FTM, SCR, UNL, WTM.)
RFP

Round Floating-point

Function       Round the contents of the extended floating-point accumulator to a normalized standard floating-point number.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>RFP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>Blank</td>
</tr>
</tbody>
</table>

Execution

The RFP instruction rounds the contents of the floating-point argument. The rounded result is the normalized standard floating-point number nearest to the value of the extended floating-point number. The floating point accumulator extension is cleared.

FOVR
- if floating-point overflow occurs at any time during the execution of the instruction FOVR is set
- otherwise it will remain unchanged.

C
- is not used and will be left clear.

V
- will remain unchanged.

Modification
- This statement has no M-field.

Machine Code Listing
136 4 0 0

Notes
1. Rounding is performed giving a result with the value nearest that of the extended floating-point number. Where there are two such results, either may be taken as the rounded result.
**RRQ**

**PLAN 2,3,4**

Read or Replace Request Block

**Function**

Read the request block stored by Executive into the program area; or replace the request block stored by Executive by the contents of specified locations in the program area.

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>RRQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   
   (a) A symbolic name referring to a lower data location.
   
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   
   (a) A decimal integer in the range 0 to 4095.
   
   (b) An octal integer in the range #0 to #7777.
   
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of 14 or 16 consecutive locations.

**Execution**

The RRQ instruction has two variants, depending upon the setting of the X field.

- **X = 0**
  
  Read the request block stored by Executive into the 16 word area starting at N(M).

- **X = 1**
  
  Replace the request block stored by Executive by the contents of the 14 or 16 word area starting at N(M). Subject to the qualifications mentioned in the Notes below, any change in the program name, introduction of new program members, or change in the priority of any program member will be implemented immediately by Executive; other changes will be recorded but not implemented, so that they will only be noticeable through subsequent RRQ (X=0) instructions or dumps of the program by Executive.

C is not used and will be left clear.

V is not used and will be left unchanged.

**Modification**

This statement has an M-field. When modified, the least significant 15 bits of N+M are taken as the operand. In the extended data mode, the least significant 22 bits of N+M are taken as the operand.

**Machine Code Listing**

166 X M N

**Notes**

1. The effect of an RRQ (X = 1) instruction is undefined if:
   
   (a) it is issued by a member of the program other than member 0.
   
   (b) when it is issued by member 0, any other member of the program is in a state other than state SL.
      
      (A member in state SL is suspended awaiting initial activation.)

2. The request block read into the object program area by an RRQ (X = 0) instruction is that which was stored by the last RRQ (X = 1) instruction, or, if no such instruction has been obeyed, that which was read when the program was loaded; except that in this latter case bit 0 of word 2 of the request block will be cleared to zero, and the checksum adjusted if appropriate. The core usage, peripheral usage and/or
priorities indicated by the request block read in by the RRQ (X = 0) instruction may differ from those currently pertaining, due to the use of GIVE with N(M) ≠ 4, ALLOT or REL instructions within the program, or to operator action; or to an earlier RRQ (R = 1).

3 The request block is read by the RRQ (X = 0) order into the 16 word area starting at N(M); if the request block is only 14 words long, then the contents of the last two words of the area after the instruction has been obeyed are indeterminate.

4 The reservation violation check on an RRQ instruction, with X = 0 or X = 1, is on the basis of a 16 word transfer, whether the request block itself is 16 or 14 words long.

5 The RRQ (X = 1) instruction may replace a 14 word request block by a 16 word request block, or vice versa, or may replace a request block by another of the same length.

6 For details of the composition of request blocks please refer to the section headed 'Binary Programs' in Chapter 7 of the Central Processors manual. The contents of the area transferred by an RRQ (X = 1) instruction must conform to the format of request blocks as therein specified, including the provision of a correct checksum. The contents of the checksum word should be such that if all the words in the block (14 or 16 words, as specified by word 0) are summed using the SUM instruction, the result is zero.

7 RRQ instructions may be used
   (a) to effect immediately:
      (i) a change of program name,
      (ii) the introduction of new program members,
      (iii) a change in the priority of any program member,
          subject to the following qualifications:
      (i) the program name must start with a letter and must consist of alphanumeric characters,
      (ii) the program name must not be the same as that of any other program currently in the machine, nor must it be 'EXEC';
   (b) to alter:
      (i) the account code (program name extension),
      (ii) the peripheral requests,
      (iii) the core store request,
          contained in the stored request block. Although such alterations are copied across to the request block store, no changes are made to current peripheral assignments or core store reservations, nor is the possibility of satisfying the requests in the present environment checked; similarly, changes in the account code, if present, are not monitored.

RRQ instructions must not be used
   (a) to delete a program member,
   (b) to set bit 0 of word 2 of the request block to 1,
   (c) to set any of bits 2 to 5 of word 2 of the request block that are not already set.

8 The RRQ instruction must not be used with any value for X other than 0 or 1.

9 Any breach of the restrictions and qualifications mentioned in notes 5, 6 and 7 has undefined effect, and may cause the program to go illegal; except that if the program is being run under the control of an operating system, and an attempt is made to change the program name to the name of another program currently in the machine, the operating system may intervene.

10 Use of the RRQ (X = 1) instruction to change the program name results in a console message of the form
   (a) with a dual programming Executive:
       #newname; WAS #oldname
   (b) with a multiprogramming Executive:
       #newname; #oldname
No console message is produced by a single programming Executive.

11 When used in machines with 4K core store or in machines without a console typewriter the RRQ instruction has null effect, except that C is left clear.

12 The RRQ instruction cannot be used to read or replace a supplementary request block. If it is required to interrogate or change the settings of address mode or branch mode the GIVE instructions with N(M)=8 or N(M)=9 respectively must be used.

13 It must be noted that certain locations are reserved for use by Executive when a program consists of two or more subprograms or members. See the section 'Locations that are Reserved in Subprogramming' on page 10 of Chapter 1 for details.
**Store Accumulator Extension**

**Function**
Store in $N(M)$ to $N + 1(M)$ the floating-point accumulator extension.

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>$N(M)$</td>
</tr>
</tbody>
</table>

$N$ may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand $N(M)$ refers to the first of two consecutive locations in which the floating-point accumulator extension is to be stored.

**Execution**

The SAE instruction clears a double-length location $N(M)$ and $N + 1(M)$ and will load the area with the contents of the floating-point accumulator extension. The contents of the floating-point accumulator extension remain unchanged.

- **FOVR** remains unchanged.
- **C** is not used and is left clear.
- **V** remains unchanged.

**Modification**
This statement has an M-field. When modified, the least significant 15 bits of $N + M$ are taken as the operand. In extended data mode, the least significant 22 bits of $N + M$ are taken as the operand.

**Machine Code Listing**

```
137 2 M N
```
Store Accumulator Extension and Zeroize

**Plan 4**  
(#XPLT ONLY)

**Function**  
Store the contents of the floating-point accumulator extension in N(M) to N + 1(M) and zeroize the floating-point accumulator extension.

**Format**  
Operation Code SAEZ  
Accumulator Blank  
Operand N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of two consecutive locations in which the floating-point accumulator is to be stored.

**Execution**

The SAEZ instruction stores the contents of the floating-point accumulator extension in the double-length location N(M) and N + 1(M). The contents of the floating-point accumulator extension are cleared.

FOVR remains unchanged.

C is not used and will be left clear.

V remains unchanged.

**Modification**  
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Machine Code Listing**

```
137 3  M  N
```
SBN

Subtract Direct Operand from X

Function
Subtract the value quoted in the operand field from the contents of X.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.
2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the value to be subtracted from X.

Execution
The SBN instruction subtracts the value of N(M) from the contents of X. If C was left set by the previous
instruction, 1 will be subtracted from the result in X. The contents of N(M) are unaltered.

If the operand is the symbolic name of a location previously defined under a #LOWER directive, the
address of that location will be subtracted from X. If the operand is written as a symbolic identifier
previously set by #DEFINE or #SET directives, the value that was assigned to that identifier is sub-
tracted from X.

C is used and will be left clear.

V will be set if overflow occurs.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M
are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as
the operand.

Machine Code Listing
103 X M N

Notes
1 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to
the symbol (not PLAN 1).
SBNC

Subtract Direct Operand from X with Carry

Function Subtract the value of the operand from the contents of X, setting C if carry occurs.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SBNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) specifies the value to be subtracted from X.

Execution

The SBNC instruction subtracts the value of N(M) from the contents of X. If C was left set by the previous instruction, At the conclusion of this instruction, B0 of the result will always be zero.

If the operand is the symbolic name of a location previously defined under a #LOWER directive, then the address of that location will be subtracted from X. If the operand is written as a symbolic identifier previously set by #DEFINE or #SET directives, the value that was assigned to that identifier is subtracted from X.

C is used and will be set if carry occurs. See Chapter 1, page 6.

V is not used and remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

In the following example, 437 is to be subtracted from a triple-length number held in X4, X5 and X6.

```
| LABEL | OPERATION | ACC   | 32 | 31 | 30 | 24 | 25 | 26 | 27 | 28 | 29 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |
|-------|-----------|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| SBNC  | 6         | 437   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| SBNC  | 5         | 0     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| SBNC  | 4         | 0     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
```

Machine Code Listing

107 XM N

Notes

1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
Subtract from Store

Function
Subtract the contents of X from the contents of N(M).
Subtract the contents of XX* from the contents of N(M) and N + 1(M).

Format
Operation Code: SBS
Accumulator: X or XX*
Operand: N(M)

If X is specified, N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

If X is specified, the operand N(M) refers to the location from which the contents of X are to be subtracted.

If XX* is specified, N may be types 1 and 2 above. When N is type 2 it should be in the range 0 to 4094. The operand N(M) refers to the first of two consecutive locations from which XX* are to be subtracted.

Execution
If X is specified, the SBS instruction subtracts the contents of X from the contents of N(M). If C was left set by the previous instruction, 1 is subtracted from the result in N(M). The contents of X are unaltered.

If XX* is specified, the double-length contents of XX* are subtracted from the double-length number in storage locations N(M) and N + 1(M). If C was left by the previous instruction, 1 is subtracted from the contents of N(M) and N + 1(M). The contents of XX* are unaltered.

C used and will be left clear.
V will be set if overflow occurs.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing
When X is specified:
013 X M N

When XX* is specified:
017 X* M N + 1
013 X M N

Notes
1. If X is specified and operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

   If XX* is specified and operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data (the symbol is given the value of the first location). Not PLAN 1.
2  SBS  XX* N(M)
    is equivalent to
    SBSC  X* N + 1(M)
    SBS  X  N(M)
SBSC

Subtract from Store with Carry

Function
Subtract the contents of X from the contents of N(M), setting C if carry occurs.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SBSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the location from which the contents of X are to be subtracted.

Execution

The SBSC instruction causes the contents of X to be subtracted from the contents of N(M). If C was left set by the previous instruction, 1 is subtracted from the result in N(M). The contents of X are unaltered. At the conclusion of this instruction, B0 of the result will always be zero.

C is used and will be set if carry occurs. See Chapter 1, page 6.

V is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

In the following example, a triple-length number in X5, X6 and X7 is to be subtracted from a number in MAKE, MAKE + 1 and MAKE + 2.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBS</td>
<td>7</td>
<td>MAKE +2</td>
<td></td>
</tr>
<tr>
<td>SBS</td>
<td>56</td>
<td>MAKE</td>
<td></td>
</tr>
</tbody>
</table>

Machine Code Listing

017 X M N

Notes

1. When dealing with double-length numbers, it is permissible to use the double-length version of the SBS instruction, but when working with numbers of greater size the SBSC instruction should be used for all but the most significant two words of the operands.

2. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
(Single-length) PLAN 1,2,3
(Double-length) PLAN 3

Function
Subtract the contents of N(M) from the contents of X.
Subtract the contents of N(M) and N + 1(M) from the contents of XX*.

Format
Operation Code  SBX
Accumulator  X or XX*
Operand  N(M)

If X is specified, N may be:
1  A relative expression which refers to a lower data location, e.g.
   (a)  A symbolic name referring to a lower data location.
   (b)  A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.
2  An absolute expression in the range 0 to 4095, e.g.
   (a)  A decimal integer in the range 0 to 4095.
   (b)  An octal integer in the range #0 to #7777.
   (c)  A previously defined absolute symbol in the range 0 to 4095.
3  A literal (not PLAN 1).

The operand N(M) refers to a location whose contents are to be subtracted
from the contents if X.

If XX* is specified, N may be types 1 and 2 above. When N is type 2 it should
be in the range 0 to 4094. The operand N(M) refers to the first of two
consecutive locations whose contents are to be subtracted from XX*.

Execution
If X is specified, the contents of N(M) are subtracted from the contents of X. If C was left set by the previous
instruction, 1 is subtracted from the result in X. The contents of N(M) are unaltered.

If XX* is specified, the contents of N(M) and N + 1(M) are subtracted from XX*. If C was left set by the
previous instruction, 1 is subtracted from the result in XX*. The contents of N(M) and N + 1(M) are unaltered.
C  is used and will be left clear.
V  will be set if overflow occurs.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as
the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing
When X is specified:  003  X  M  N
When XX* is specified:  007  X  M  N + 1
                        003  X  M  N
Notes

1 If X is specified and operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

If XX* is specified and operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data (the symbol is given the value of the first location). Not PLAN 1.

2 SBX XX* N(M)
   is equivalent to
   SBXC X* N + 1(M)
   SBX X N(M)
SBXC

Function
Subtract the contents of N(M) from the contents of X, setting C if carry occurs.

Format
Operation Code
SBXC

Accumulator
X

Operand
N(M)

N may be:
1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
decimal or octal integer.

2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

3 A literal (not PLAN 1).

The operand N(M) refers to the location the contents of which are to be
subtracted from the contents of X.

Execution

The SBXC instruction subtracts the contents of N(M) from the contents of X. If C was left set by the
previous instruction, 1 is subtracted from the result of X. At the conclusion of this instruction, B0 of
the result will always be zero. The contents of N(M) are unaltered.

C is used and will be set if carry occurs. See Chapter 1, page 6.

V is not used and remains unchanged.

Modification 
This statement has an M-field. When modified, the least significant 15 bits of N + M are
taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the
operand.

Example
In the following example, a triple-length number held in LIAM, LIAM + 1 and LIAM + 2 is to be sub-
tracted from the contents of X4, X5 and X6.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>N</th>
<th>M</th>
<th>X</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBXC</td>
<td>6</td>
<td>LIAM+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBX</td>
<td>4,5</td>
<td>LIAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Machine Code Listing
007 X M N

Notes
1 When dealing with double-length numbers in PLAN 3 it is possible to use the double-length version
of the SBX instruction, but when working with numbers of greater size, the SBXC instruction should
be used for all but the most significant two words of the operands.

2 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to
the symbol (not PLAN 1).
**SCR**

**Function**
Open a magnetic tape as a work tape.

**Format**
- **Operation Code**: SCR
- **Accumulator**: A decimal digit in the range 0 to 7 specifying the program's unit number for the magnetic tape unit.
- **Operand**: Blank

**Execution**
SCR causes a magnetic tape with an expired retention period to be opened and relabelled 'SCRATCH TAPE', with a retention period of zero.

**Modification**
The SCR macro-instruction has no M-field.

**Machine Code Listing**
157 X 0 n LT

when n is the relative address in the literals table of the first of two consecutive locations which hold a control area of the form:

- first word: 5 / #600
- second word: 0

**Notes**
1. This is one of a set of magnetic tape macro-instructions which permit the execution of simple basic functions on magnetic tape without requiring the programmer to set up control areas. (See also BSP, BTM, CLOSE, FTM, REW, UNL, WTM.)
PLANT 4 (#XPLT ONLY)
(Macro-instruction)

Function
Store the contents of the extended floating-point accumulator in N(M) to N + 3(M).

Format
Operation code: SEFP
Accumulator: Blank
Operand: N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of four consecutive locations that hold the extended floating-point number.

Execution
The SEFP instruction will generate two machine code instructions. The effect of the instruction is to store the contents of the accumulator extension in the double-length location N + 2(M) and N + 3(M). The contents of the floating-point accumulator are then stored in the double-length location N(M) and N + 1(M). The contents of the extended floating-point accumulator remain unchanged.

FOVR remains unchanged.
C is not used and will be left clear.
V will be set if FOVR is set when the instruction is about to be executed otherwise it remains unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing
137 2 M N + 2
137 0 M N

Notes
SEFP N(M) is equivalent to
SAE N(M)
SFP N(M)
SEFPZ  

Store Extended Floating-Point and Zeroize  

Function  
Store the contents of the extended floating-point accumulator in N(M) to N + 3(M) and zeroize the extended floating-point accumulator.

Format  

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SEFPZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operator</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of four consecutive locations that hold the extended floating-point number.

Execution  

The SEFPZ instruction will generate two machine code instructions. The effect of the instruction is to store the contents of the floating-point accumulator extension in the double-length location N + 2(M) and N + 3(M). The contents of the floating-point accumulator extension are cleared. The instruction will then store the contents of the floating-point accumulator in the double-length location N(M) and N + 1(M). The contents of the floating-point accumulator are cleared.

\[ \text{FOVR} \] will be left clear  
\[ C \] is not used and will be left clear  
\[ V \] will be set if \[ \text{FOVR} \] is set when the instruction is about to be executed otherwise it will remain unchanged.

Modification  
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing  

- 137 3 M N  
- 137 1 M N

Notes  
SEFPZ N(M) is equivalent to  
SAEZ N(M)  
SFPZ N(M)
SFP

PLAN 1,2,3

Function  Store in N(M) and N + 1(M) the floating-point number held in the floating-point accumulator A.

Format  

Operation Code  SFP
Accumulator  Blank
Operand  N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of two consecutive locations in which the floating-point number is to be stored.

Execution
The SFP instruction clears the double-length location N(M) and N + 1(M) and will load the area with the floating-point number in A. The contents of A are unaltered.

FOVR remains unaltered.

C is not used and will be left clear.

V is used and will be set if FOVR is set when the instruction comes to be obeyed.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Examples

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SFP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SFP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>WARDY+1</td>
</tr>
</tbody>
</table>

The first example above stores the floating-point number held in the floating-point register A in the double-length location WARDY and WARDY + 1. The second example stores the floating-point number held in the floating-point register A in the double-length location STEF + 3 and STEF + 4.

Machine Code Listing

137 0 M N

Notes
1. If operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data. (The symbol is given the value of the first location.)
SFPZ

Store Floating-point and Zeroize

Function
Load the contents of the floating-point accumulator A into the store locations N(M) and N + 1(M), and leave A clear.

Format
Operation Code  SFPZ
Accumulator     Blank
Operand         N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range \#0 to \#7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand refers to the first of two consecutive locations which are to be loaded with the contents of the floating-point accumulator.

Execution
The SFPZ instruction loads the contents of the floating-point accumulator A into store locations N(M) and N + 1(M). Accumulator A, FOVR and the exponent register are left clear.

FOVR will be left clear.

C is not used and will be left clear.

V is used and will be set if FOVR was set when the instruction came to be obeyed.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Examples

| LABEL | OPERATION | ACC | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|-------|------------|-----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1     | SFPZ       |     |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 2     | SFPZ       |     |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

The first example above stores the floating-point number held in the floating-point accumulator A in the double-length location PAUD and PAUD + 1. Register A and the exponent Overflow register are left clear.

Machine Code Listing

137 1 M N

Notes

1. If operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data. (The symbol is given the value of the first location.)
SLA

Function
The contents of X or XX* are arithmetically shifted to the left the number of places specified by N(M).

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X or XX*</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
- An absolute expression in the range 0 to 1023, e.g.
  (a) A decimal integer in the range 0 to 1023.
  (b) An octal integer in the range #0 to #1777.
  (c) A previously defined absolute symbol with a value in the range 0 to 1023.

The operand N(M) specifies the number of positions to be shifted.
If X is specified N(M) should not exceed 24.
If XX* is specified N(M) should not exceed 47.

Execution
When X is specified, the SLA instruction regards the contents of the accumulator as a signed single-length number. Bits 0 to 23 of accumulator X are shifted to the left the number of places specified in N(M). Zeros are propagated at the least significant end of X in the spaces created by the shift.

When XX* is specified, the contents of the accumulator are regarded as a signed double-length number. The sign bit (B0) of X* is left unchanged if N(M) = 0, otherwise it is cleared, bits being carried from BI of X* to B23 of X. B0 to B23 of X, BI to B23 of X* are shifted to the left the number of places specified in N(M). Zeros are propagated at the least significant end of X* in the spaces created by the shift. If the shift is for more than 23 positions, zeros will also be propagated at the least significant end of X.

C is not used and will be left clear.

V will be set if B0 of X undergoes any change during shifting (i.e. if the sign of the number changes).

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Examples
1 Original contents of X5:  101 110 111 001 010 100 111 011

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Code</th>
<th>N</th>
<th>M</th>
<th>N</th>
<th>M</th>
<th>N</th>
<th>M</th>
<th>N</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLA</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation:  SLA 5 3

Result in X5:  110 111 001 010 100 111 011 000

State of V:  Set because the original B1, a zero, entered the B0 position and was different from the original setting of B0.

432017 771

Chapter 4 201
Original contents of X5: 111 110 001 010 100 111 001 101

<table>
<thead>
<tr>
<th>Operation:</th>
<th>SLA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Result in X5:</td>
<td>110 001 010 100 111 001 101 000</td>
<td></td>
</tr>
<tr>
<td>State of V:</td>
<td>Unset as the answer is arithmetically correct. (B0 has not undergone any change during the operation.)</td>
<td></td>
</tr>
</tbody>
</table>

Machine Code Listing

When X is specified: 110 X M N
When XX* is specified: 111 X M N

Notes

1 To distinguish these instructions from other shift instructions the compiler makes the two most significant bits of the 12-bit address equal to two. If modification affects these bits, the type of shift will be altered.

2 The SLA instruction may be used to multiply negative or positive numbers by powers of two because each shift left of one position doubles the value of the number.
**PLAN 1,2,3**  

**Shift Left Circular**

**Function**  
The contents of X or XX* are circulated to the left the number of places specified by N(M).

**Format**  

*Operation Code*: SLC  

*Accumulator*: X or XX*  

*Operand*: N(M)

N may be:  
- An absolute expression in the range 0 to 1023, e.g.  
- A decimal integer in the range 0 to 1023.  
- An octal integer in the range #0 to #1777.  
- A previously defined absolute symbol with a value in the range 0 to 1023.

The operand N(M) specifies the number of positions to be shifted.  
If X is specified N(M) should not exceed 24.  
If XX* is specified N(M) should not exceed 48.

**Execution**  
When X is specified, the SLC instruction regards the contents of the accumulator as a 24-bit pattern. The pattern is circulated to the left the number of places specified in the operand. As a bit 'leaves' the most significant end of X, it 're-appears' in the space simultaneously created at the least significant end.

When XX* is specified, the contents of the accumulators are regarded as a 48-bit pattern. The pattern is circulated to the left the number of places specified in the operand. As a bit 'leaves' the most significant end of X, it 'appears' in the space simultaneously created at the least significant end of X*.

C is not used and will be left clear.

V is not used and remains unchanged.

**Modification**  
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Examples**

1. Original contents of X3: 110 010 111 000 101 011 110 101

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>12</td>
<td>01</td>
</tr>
</tbody>
</table>

   Operation: SLC 3 6

   Result in X3: 111 000 101 011 110 101 110 010

2. X5 contains four 6-bit characters. It is required to place the N3 character in the N2 position without losing the remaining three characters.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>12</td>
<td>01</td>
</tr>
</tbody>
</table>

   Operation: SLC 5 6

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>12</td>
<td>01</td>
</tr>
</tbody>
</table>
Machine Code Listing

When X is specified:  110 X M N
When XX* is specified:  111 X M N

Notes

1. To distinguish this instruction from other shift instructions the compiler makes the two most significant bits of the 12-bit address equal to 0. If the modifier affects these bits, the type of shift will be altered.

2. The circular shifts may be used in placing characters into required positions of a word without losing the rest of the word.
Plan 1, 2, 3

**Function**
The contents of X or XX* are moved to the left the number of places specified by N(M).

**Format**
- **Operation Code**: SLL
- **Accumulator**: X or XX*
- **Operand**: N(M)

N may be:
- An absolute expression in the range 0 to 1023, e.g.
  - (a) A decimal integer in the range 0 to 1023.
  - (b) An octal integer in the range #0 to #1777.
  - (c) A previously defined absolute symbol with a value in the range 0 to 1023.

The operand N(M) specifies the number of positions to be shifted.
- If X is specified N(M) should not exceed 24.
- If XX* is specified N(M) should not exceed 48.

**Execution**
When X is specified, the SLL instruction regards the contents of the accumulator as a 24-bit pattern. The pattern is shifted to the left the number of positions specified in the operand, and any bits which overflow at the most significant end of the word are lost. Spaces created at the least significant end of the word are filled with zeros.

When XX* is specified, the SLL instruction regards the contents of the accumulators as a 48-bit pattern. The pattern is shifted to the left the number of positions specified in the operand, and any bits which overflow at the most significant end of X are lost. Spaces created at the least significant end of X* are filled with zeros. If the shift is for more than 23 positions, spaces created at the least significant end of X are also filled with zeros.

C is not used and will be left clear.

V is not used and remains unchanged.

**Modification**
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Examples**

1. **Original contents of X3**: 111 010 101 110 011 101 110 001

   ![Operation Diagram](image)

   **Operation**: SLL 3.

   **Result in X3**: 101 011 100 111 011 100 010 000

2. X3 contains a character in the n3 position. It is required to place this character in the n0 position, and to clear n1, n2 and n3.

   ![Operation Diagram](image)
Machine Code Listing

When X is specified: 110 X M N
When XX* is specified: 111 X M N

Notes

1 To distinguish this instruction from other shift instructions the compiler makes the two most significant bits of the 12-bit address equal to 1. If modification affects these bits, the type of shift will be altered.
PLAN 3.4  

**Supplementary Modifier to Operand of Next Instruction**

*(Available on 1902A, 1903A and Processors with Extended Data Mode Facilities)*

**Function**  
Cause the least significant 15 bits (compact mode) or 22 bits (extended data mode) of the content of N(M) to be used as a modifier in the next instruction.

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   - (a) A decimal integer in the range 0 to 4095.
   - (b) An octal integer in the range #0 to #7777.
   - (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the location which contains the supplementary modifier.

**Execution**

The SMO instruction causes the least significant 15 bits (compact mode) or 22 bits (extended data mode) of the word whose address is specified by N(M) to be used as a modifier in the next instruction. This SMO modifier will supplement any modifier which the instruction may already be using. It is illegal to have a SMO instruction immediately following a SMO instruction or for an OBEY instruction ultimately to address a SMO if it is itself modified by means of a SMO instruction. Otherwise, any instruction can be modified using SMO, although there may be no effect (e.g. NULL). If a SMO instruction precedes an OBEY instruction, the modification is applied to the OBEY instruction, not to the instruction obeyed.

C is not used and will be left unchanged.

V is not used and remains unchanged.

**Modification**  
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Example**

If X3 contains the integer 7 and location SCRUM contains 8, then the following series of instructions will cause the contents of TEAM + 15 to be loaded into X2:

```
| Label | Operation | ACC 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------|-----------|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| SMO   | SCRUM     | TEAM (3)|    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
| LDX   | 2         |        |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |
```

**Machine Code Listing**

117 0 M N

**Notes**

1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
SRA

Shift Right Arithmetic

Function
The contents of X or XX* are arithmetically shifted to the right the number of places specified in N(M).

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X or XX*</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
An absolute expression in the range 0 to 1023, e.g.
(a) A decimal integer in the range 0 to 1023.
(b) An octal integer in the range #0 to #1777.
(c) A previously defined absolute symbol with a value in the range 0 to 1023.

The operand N(M) specifies the number of positions to be shifted.
If X is specified N(M) should not exceed 24.
If XX* is specified N(M) should not exceed 47.

Execution
When X is specified, the SRA instruction regards the contents of the accumulator as a signed single-length number. Bits 0 to 23 of accumulator X are shifted to the right the number of places specified in N(M). B0 (the sign bit) is propagated at the most significant end of the word, thus presenting an arithmetically correct answer. Bits shifted beyond B23 are lost except that the value of the final bit so shifted is added back into the result to round off the answer.

When XX* is specified, the contents of the accumulator are regarded as a signed double-length number. The sign bit B0 of X* is left unchanged if N(M) = 0, otherwise it is cleared, bits being carried from B23 of X to B1 of X*. B0 to B23 of X and B1 to B23 of X* are shifted to the right the number of places specified in N(M). B0 (the sign bit) is propagated at the most significant end of X (and X* in the case of a shift of more than 23 places), thus presenting an arithmetically correct answer. Bits shifted beyond B23 of X* are lost and there is no rounding.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Examples
1 Original contents of X5: 111 010 111 001 100 111 000 111

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>13 12</th>
<th>11 10</th>
<th>9 8</th>
<th>7 6</th>
<th>5 4</th>
<th>3 2</th>
<th>1 0</th>
<th>OPERAND</th>
<th>31 30</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SRA</td>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation:

Result in X5: 111 111 010 111 001 100 111 001

Chapter 4
2 Original contents of X4: 011 010 111 001 100 111 000 011

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>N</th>
<th>M</th>
<th>L</th>
<th>O</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SRA</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation: SRA

Result in X4: 000 011 010 111 001 100 111 000

3 Original contents of X6: 011 010 111 001 100 111 000 111

and of X7:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>N</th>
<th>M</th>
<th>L</th>
<th>O</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SRA</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation: SRA

Result in X6: 000 011 010 111 001 100 111 000

Result in X7: 011 111 010 111 001 100 111 000

Machine Code Listing

When X is specified: 112 X M N
When XX* is specified: 113 X M N

Notes

1 To distinguish these instructions from other shift instructions, the compiler makes the two most significant bits of the 12-bit address equal to two. If modification affects these bits, the types of shift will be altered.

2 The SRA instruction may be used to divide numbers by powers of two, because each shift of one place halves the value of the number.
SRAV

Special Shift Right

Function
The contents of X or XX* are arithmetically shifted to the right the number of places specified in N(M).

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SRAV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X or XX*</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
- An absolute expression in the range 0 to 1023, e.g.
- A decimal integer in the range 0 to 1023.
- An octal integer in the range 0 to 1777.
- A previously defined absolute symbol with a value in the range 0 to 1023.

The operand N(M) specifies the number of positions to be shifted.
If X is specified N(M) should not exceed 24.
If XX* is specified N(M) should not exceed 47.

Execution

When V is clear, the SRAV instruction has exactly the same effect as the SRA instruction.

When V is set and X is specified, the SRAV instruction clears V (but see below). Bits 0 to 23 of accumulator X are shifted to the right the number of places specified in N(M). The inverse of the sign bit (B0) is propagated at the most significant end of the word. Bits shifted beyond B23 are lost except that the value of the final bit so shifted is added back into the result to round it off.

When V is set and XX* is specified, the SRAV instruction clears V (but see below). B0 of X* is left unchanged if N(M)=0, otherwise it is cleared, bits being carried from B23 of X to B1 of X*. B0 to B23 of X and B1 to B23 of X* are shifted to the right the number of places specified in N(M). The inverse of the sign bit (B0 of X) is propagated at the most significant end of X (and of X* in the case of a shift of more than 23 places). Bits shifted beyond B23 of X* are lost and there is no rounding.

C is not used and will be left clear.

V is used and will be left clear, except that if the least significant ten bits of N + M are zero, V will be left unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

Original contents of X3: 011 001 111 010 100 110 001 111

<table>
<thead>
<tr>
<th>LABEL</th>
<th>4</th>
<th>7</th>
<th>0</th>
<th>3</th>
<th>6</th>
<th>9</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERATE</td>
<td>SLA</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Operation 1:

Result in X3: 110 011 110 101 001 100 011 110

State of Overflow register: Set

210
Operation 2:

Result in X3: 011 001 111 010 100 110 001 111

State of Overflow register: Clear

The SLA instruction caused V to be set, because the sign bit of the word was changed. The SRAV instruction was then used to recover the original number.

**Machine Code Listing**

When X is specified: 112 X M N
When XX* is specified: 113 X M N

**Notes**

1. To distinguish these instructions from other shift instructions, the compiler makes the two most significant bits of the 12-bit address equal to 5. If modification affects these bits, the type of shift will be altered.

2. If there is a possibility of a one-bit overflow as a result of a V-setting addition, subtraction or arithmetic shift left instruction, then the SRAV instruction may be used to recover the value of the overflowed bit.

3. If the SRAV instruction with X specified is given when V is set, and N(M)=1 and X contains #77777777, then the rounding will give an arithmetically incorrect result (#40000000 in X) but V will be left clear.
 SRC

Shift Right Circular

Function  The contents of X or XX* are circulated to the right the number of places specified by 
N(M).

Format  Operation Code  SRC
 Accumulator  X or XX*
 Operand  N(M)

N may be:
An absolute expression in the range 0 to 1023, e.g.
(a) A decimal integer in the range 0 to 1023.
(b) An octal integer in the range #0 to #1777.
(c) A previously defined absolute symbol with a value in the range 
0 to 1023.

The operand N(M) specifies the number of positions to be shifted.
If X is specified N(M) should not exceed 24.
If XX* is specified N(M) should not exceed 48.

Execution
When X is specified, the SRC instruction regards the contents of the accumulator as a 24-bit pattern.
The pattern is circulated to the right the number of places specified in the operand. As a bit 'leaves'
the least significant end of X, it 'reappears' in the space simultaneously created at the most significant 
end.

When XX* is specified, the contents of the accumulators are regarded as a 48-bit pattern. The pattern 
is circulated to the right the number of places specified in the operand. As a bit 'leaves' the least 
significant end of X*, it 'reappears' in the space simultaneously created at the most significant end of X.

C  is not used and will be left clear.

V  is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are 
taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the 
operand.

Examples
1  Original contents of X3:  110 010 111 000 101 011 110 101

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SRC</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Result in X3:  101 011 110 110 110 010 111 000

2  X5 contains four 6-bit characters. It is required to place the n1 character in the n3 position without
losing the remaining three characters.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SRC</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Machine Code Listing

When X is specified:  112 X M N
When XX* is specified:  113 X M N

Notes

1. To distinguish this instruction from other shift instructions the compiler makes the two most significant bits of the 12-bit address equal to 0. If modification affects these bits, the type of shift will be altered.

2. The circular shifts may be used to reposition data within the word (or words) without losing information.
SRL

Shift Right Logical

Function
- The contents of X or XX* are moved to the right the number of places specified in N(M).

Format
- **Operation Code**: SRL
- **Accumulator**: X or XX*
- **Operand**: N(M)

N may be:
- An absolute expression in the range 0 to 1023, e.g.
  - (a) A decimal integer in the range 0 to 1023.
  - (b) An octal integer in the range 0 to 7777.
  - (c) A previously defined absolute symbol with a value in the range 0 to 1023.

The operand N(M) specifies the number of positions to be shifted.
- If X is specified N(M) should not exceed 24.
- If XX* is specified N(M) should not exceed 48.

Execution
- When X is specified, the SRL instruction regards the contents of the accumulator as a 24-bit pattern. The pattern is shifted to the right the number of positions specified in the operand. Any bits overflowing at the least significant end of the word are lost. Spaces created at the other end of the word are filled with zeros.
- When XX* is specified, the contents of the accumulators are regarded as a 48-bit pattern. The pattern is shifted to the right the number of positions specified in the operand. Any bits overflowing at the least significant end of X* are lost. Spaces created at the most significant end of X (and X* if more than 23 positions are shifted) are filled with zeros.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
- This statement has an M-field. When modified, the least significant 15 bits of \( N + M \) are taken as the operand. In the extended data mode, the least significant 22 bits of \( N + M \) are taken as the operand.

Examples
1. **Original contents of X2**: 101 010 111 000 101 110 111 100

   | LABEL | OPERATION | ACC. | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N11 | N12 | N13 | N14 | N15 | N16 | N17 | N18 | N19 | N20 | N21 | N22 | N23 |
   |-------|-----------|------|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
   | SRL   | 2         |      |    |    |    |    |    |    |    |    |    |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

   Result in X2: 000 001 010 101 110 001 011 101

2. **X5 contains a character in the n1 position. It is required to place this character in the n3 position, and to clear n0 and n1.**

   | LABEL | OPERATION | ACC. | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N11 | N12 | N13 | N14 | N15 | N16 | N17 | N18 | N19 | N20 | N21 | N22 | N23 |
   |-------|-----------|------|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
   | SRL   | 5         |      |    |    |    |    |    |    |    |    |    |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

   | SRL   | 12        |      |    |    |    |    |    |    |    |    |    |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

214 Chapter 4
Machine Code Listing

When X is specified: 112 X M N
When XX* is specified: 113 X M N

Notes

1 To distinguish these instructions from other shift instructions the compiler makes the two most significant bits of the 12-bit address equal to 1. If modification affects these bits, the type of shift will be altered.
STO

Store the Contents of X

Function
Store the contents of X in N(M).
Store the contents of XX* in N(M) and N + 1(M).

Format

Operation Code  STO

Accumulator  X or XX*

Operand  N(M)

If X is specified, N may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed
       decimal or octal integer.

2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

If X is specified, the operand N(M) refers to the location in which the contents
of X are to be stored.

If XX* is specified, N may be types 1 and 2 above. When N is type 2 it should
be in the range 0 to 4094. The operand N(M) refers to the first of two
consecutive locations in which the contents of XX* are to be stored.

Execution
If X is specified, the contents of X are stored in N(M): if C was left set by the previous instruction, 1 is added to
the result in N(M). The contents of X are unaltered.

If XX* is specified, the contents of XX* are loaded into N(M) and N + 1(M): 1 is added to N + 1(M) if C was left
set by the previous instruction. The contents of XX* are unaltered.

C is used and will be left clear.

V will be set if X (single-length) or X* (double-length) contains #37777777 and C was set initially.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken
as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing
When X is specified:  010  X  M  N
When XX* is specified:  010  X*  M  N + 1
                                010  X  M  N

Notes
1 If X is specified and operand type 1(a) is an undefined symbol the compiler will allocate a location in
  lower data to the symbol (not PLAN 1).
If XX* is specified and operand type 1(a) is an undefined symbol the compiler will allocate two consecutive locations in lower data (the symbol is given the value of the first location); not PLAN 1.

\[ \text{STO } \text{XX* } \text{N(M)} \]

is equivalent to

\[ \text{STO } \text{x* } \text{N + 1(M)} \]

\[ \text{STO } \text{X } \text{N(M)} \]
STOC

Store the Contents of X with Carry

Function
Store the contents of X into N(M), setting C if carry occurs.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>STOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the location in which the contents of X are to be stored.

Execution
The contents of X are stored into N(M). If C was left set by the previous instruction, 1 is added to the result in N(M). The contents of X are unaltered. At the conclusion of this instruction, B0 of the result will always be zero.

C is used and will be set if carry occurs. See Chapter 1, page 6.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example
In the following example the single-length positive quantity in X0 is added to the triple-length number in X5, X6 and X7 leaving the result in MOSS, MOSS + 1, and MOSS + 2.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STC</td>
<td>0</td>
<td>MOSS+2</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>MOSS+2</td>
</tr>
<tr>
<td></td>
<td>ADSC</td>
<td>0</td>
<td>MOSS+2</td>
<td></td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>MOSS+2</td>
</tr>
<tr>
<td></td>
<td>STC</td>
<td>0</td>
<td>MOSS+1</td>
<td></td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>MOSS+1</td>
</tr>
<tr>
<td></td>
<td>STC</td>
<td>0</td>
<td>MOSS</td>
<td></td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>MOSS</td>
</tr>
</tbody>
</table>

Machine Code Listing
014 X M N

Notes
1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
PLAN 1,2,3

Function  Zeroize the word specified by N(M).

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>STOZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the location to be zeroized.

Execution

The STOZ instruction clears the storage location whose address is given by N(M).

- C is not used and will be left clear.
- V is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing

033 0 M N

Notes

1. If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).
SUM

Sum N Words

Function
Place the sum of N words starting from the address specified in X₀ into X.

Format
Operation Code    SUM
Accumulator       X
Operand            N(M)

N may be:
An absolute expression in the range 0 to 4095, e.g.
(a) A decimal integer in the range 0 to 4095.
(b) An octal integer in the range #0 to #7777.
(c) A previously defined symbol with a value in the range 0 to 4095.
The operand N(M) specifies the number of words to be summed in X, and must have a value in the range 0 to 511 inclusive.

Execution
The SUM instruction replaces the previous contents of X with the sum of N consecutive locations. The address of the first location is contained in the least significant 15 bits of X₀ (in the extended data mode, the least significant 22 bits of X₀). If the sum exceeds the capacity of one word, then bits will be lost from the more significant end but V will not be set.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example
It is required to sum the contents of 47 consecutive storage locations starting from TAX and to store the result in location TOTAL.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>OPERAND</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
<th>FROM</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDN</td>
<td>6</td>
<td>TAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LOAD</td>
<td>START</td>
</tr>
<tr>
<td>SUM</td>
<td>5</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SUM</td>
<td>47</td>
</tr>
<tr>
<td>S.T.</td>
<td>5</td>
<td>TOTAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>STORE</td>
<td>CONTENTS</td>
</tr>
</tbody>
</table>

Machine Code Listing

127  X  M  N
SUSAR

PLAN 3  

Suspend Current Program Member Awaiting Reactivation

Function
Suspend the current program, subprogram or member unless or until further action is required of it by an AUTO statement from another member.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SUSAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>1</td>
</tr>
<tr>
<td>Operand</td>
<td>Blank</td>
</tr>
</tbody>
</table>

Execution
The statement allows a program member to suspend itself awaiting reactivation by another member. If however a request for the reactivation of this current member has been 'queued' by the setting of the M indicator, then the member is not suspended, but the M indicator is cleared and the member proceeds to its next instruction.

$C$ is not used and will be left clear.

$V$ is not used and remains unchanged.

Modification
This statement has an M-field (but N(M) must be zero).

Machine Code Listing
164 1 0 0

Notes
1. On being reactivated a program member should ascertain the reason for reactivation.
2. When the member is reactivated it will be entered at the location following the last SUSAR or SUSIN instruction which it issued.
3. The effect of SUSAR with any value other than 1 in the accumulator field is undefined.
SUSBY

Suspend if Busy

Function
Suspend program member if specified peripheral is busy.

Format
Operation Code  SUSBY

The rest of the instruction can take three possible forms:

1 Accumulator  Blank
Operand
The symbolic name of the relevant peripheral (type and program's unit or file number, thus: LP0, TP2, etc.). The symbols for the permitted peripheral types are specified in Note 1.

2 Accumulator  The program's unit number of the relevant peripheral. With direct access devices, the program's file number.
Operand
N(M), where N(M) is the type number of the peripheral. The type numbers of the permitted peripherals are specified in Note 1.

3 Accumulator  X, where X is an accumulator whose least significant six bits contain the unit number of the relevant peripheral or, with direct access devices, the program's file number.
Operand
N(M), where N(M) = 256 + the type number of the peripheral as specified for 2 above.

PLAN 1 compilers will not accept operands of type 1.

Execution
The SUSBY instruction will cause Executive to test whether the unit specified by N is busy. If it is busy, the program member giving the order will be suspended until such time as the unit is free; the program member may then continue. With direct access devices the program member is suspended while any transfer specifying the same file number is incomplete.

C  is not used and will be left clear.
V  is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand. See Note 2 below.

Examples

| LABEL | OPERATION | ACC | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------|-----------|-----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1     | SUSBY     | MTA |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 2     | SUSBY     | 6   |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 3     | SUSBY     | 261 |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 4     | SUSBY     | 238 |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

In each case the program is suspended if the magnetic tape unit, MT4, is busy.

Machine Code Listing

150 X M N
Notes

1. The peripheral types are specified in the operand field in the following ways:

<table>
<thead>
<tr>
<th>Peripheral Type</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper Tape Reader</td>
<td>TR</td>
<td>0</td>
<td>256</td>
</tr>
<tr>
<td>Paper Tape Punch</td>
<td>TP</td>
<td>1</td>
<td>257</td>
</tr>
<tr>
<td>Line Printer</td>
<td>LP</td>
<td>2</td>
<td>258</td>
</tr>
<tr>
<td>Card Reader</td>
<td>CR</td>
<td>3</td>
<td>259</td>
</tr>
<tr>
<td>Card Punch</td>
<td>CP</td>
<td>4</td>
<td>260</td>
</tr>
<tr>
<td>Magnetic Tape</td>
<td>MT</td>
<td>5</td>
<td>261</td>
</tr>
<tr>
<td>Exchangeable Disc</td>
<td>ED</td>
<td>6</td>
<td>262</td>
</tr>
<tr>
<td>Magnetic Card File</td>
<td>MC</td>
<td>7</td>
<td>263</td>
</tr>
<tr>
<td>Magnetic Drum</td>
<td>DR</td>
<td>9</td>
<td>265</td>
</tr>
<tr>
<td>Cassette Tape</td>
<td>CT</td>
<td>12</td>
<td>268</td>
</tr>
<tr>
<td>Fixed Disc</td>
<td>FD</td>
<td>13</td>
<td>269</td>
</tr>
<tr>
<td>Digital Incremental Plotter</td>
<td>GP</td>
<td>20</td>
<td>276</td>
</tr>
</tbody>
</table>

2. The operand is modifiable, except in the case of a type 1 operand. If the operand is of type 2 or 3 the unit type may be modified. Note that a type 2 operand would be interpreted as a type 3 operand if B15 was set in the modifier.

3. If a type 3 operand is used, bits 0 to 17 of the accumulator specified by X must be set to zero.

4. A peripheral is regarded as busy in this context if significance has been defined for B0 of the reply word for a PERI instruction and that bit is set to one.

5. The order may be used to ensure that each transfer of data between peripheral and processor is completed before the data involved in the transfer is processed or the store area involved is used for further transfers.

6. Where a SUSBY instruction refers to a card reader fitted with Decoder E (card reader type 2102) and Executive's code translation facility is in use, the suspension is not removed until the translation process is completed. This is some time after the actual reading is completed.
SUSDP

Suspend and Dump

Function
Suspend the program, and dump the program area on the specified peripheral unit.

Format
Operation Code  SUSDP

The rest of the instruction can take three possible forms:

1  Accumulator  Blank
   Operand  The symbolic name of the relevant peripheral (CP1, TP0, etc.).

2  Accumulator  The program's unit number of the relevant peripheral.
   Operand  N(M), where N(M) is the type number of the peripheral:
           1 = Paper Tape Punch
           4 = Card Punch
           5 = Magnetic Tape

3  Accumulator  X, where X is an accumulator whose least significant six bits contain the
   Operand  program's unit number of the relevant peripheral.
           N(M), where N(M) = 256 + the type number of the peripheral as specified
           in 2 above.

PLAN 1 compilers will not accept operands of type 1.

Execution
The SUSDP instruction causes the program to be suspended immediately. Executive then dumps the
whole of the program area, in binary program form, onto the specified peripheral unit. The dump of
the program area is preceded by a request block (see Note 5 below) and terminated by a type 3 (suspend)
entry block. On completion of the dumping, the program resumes with the instruction in the location
following that which contained the SUSDP instruction.

C  is not used and will be left clear.

V  is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are
taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the
operand. See Note 1 below.

Examples

| LABEL | OPERATION | ACC. | M | N | R | H | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 |
|-------|-----------|------|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | SUSDP     | T.P0.|    |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|       | SUSDP     | 0.   |    |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|       | SUSDP     | 25.7|    |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|       | SUSDP     | 25.6|    |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

In each of these cases the program will be dumped on tape punch TP0.

Machine Code Listing
155  X M N
Notes

1. The operand is modifiable, except in the case of a type 1 operand. If the operand is of types 2 or 3, the unit type may be modified. Note that a type 2 operand would be interpreted as a type 3 operand if B15 was set in the modifier.

2. If a type 3 operand is used, bits 0 to 17 of the accumulator specified by X must be set to zero.

3. If the program consists of more than one member, the SUSDP instruction is valid only if initiated by member 0. It causes all members of the program to be suspended immediately. On completion of the dumping process all members of the program which were active at the time the SUSDP instruction was given are reactivated at the appropriate locations.

4. The instruction is illegal if:
   (a) the specified device is not of a suitable type (suitable types include Paper Tape Punches, Card Punches and Magnetic Tape; but not Cassette Tape),
   (b) the specified unit is not assigned to the program,
   (c) it is issued by a member of the program other than member 0,
   (d) on a machine without a console typewriter.

5. Unless it has been changed by the use of an RRQ instruction between the time of loading and the time the SUSDP instruction is obeyed, the request block output will be a copy of the original request block, except that bit 0 of word 2 will always be zero. The core store requirement will, unless it has been changed by an RRQ instruction, always be the original value; this may not be the actual core usage at the time of dumping if the GIVE with N(M) = 4 order has been used, or if the operator had specified a core utilization figure at the time of loading. The situation is similar with regard to peripheral requests; the peripheral usage may have been varied during the running of the program. See further under Note 12.

   If, at the time the SUSDP instruction is obeyed, member 0 of the program is operating in modes other than compact mode and direct branch mode (15AM and DBM), then a supplementary request block will be included in the dump. The operating modes at the time of dumping may differ from those pertaining at the time the program was loaded, if a GIVE with N(M) = 9 order has been used during the running of the program.

6. The dump will be in standard binary program format, in type 5 data block pairs if to magnetic tape on a 1904 processor and above, or in type 0 data blocks in all other cases. Dumps in the form of type 0 data blocks omit any blocks in which all 16 program words are zero.

7. The entry block produced is of type 3 (suspend). The entry address contained in word 1 of the block is that of the location following that from which the SUSDP instruction was obeyed.

8. The dump is initiated without any initial repositioning of the medium.

9. Should a failure occur during the dumping of the program, typing 'GO' will not cause the SUSDP instruction to be repeated, hence a major restart must be carried out in such cases.

10. When a SUSDP instruction is initiated, Executive starts the dumping process without checking the state of any peripheral devices, other than the one to be used for the dump. If, therefore, any other peripheral units are assigned to the program and are active at the time the instruction is given, the result is undefined. Further, in these circumstances an incorrect checksum may be generated, rendering reload of the dump impossible.

11. If any flag-setting peripheral is assigned to the program, the effects of a SUSDP instruction are undefined.

12. If a program is run under the control of a GEORGE 3 or GEORGE 4 operating system, any SUSDP instruction encountered is treated as illegal.

13. The dumped program will not contain details of the state of V, C and zero suppression mode at the time of the dump, nor of the suspension state of members other than member 0, nor of the associated M, P and E indicators. It follows that if the dump is subsequently reloaded, the state of V and zero suppression mode will be indeterminate, and members other than member 0 will be suspended awaiting initial activation (i.e., in state SL); the address mode and the branch mode
of member 0 will be those current at the time of the dump.

14 Information on binary program formats, and on the various kinds of request, data and entry blocks, may be found in the Central Processors manual, Chapter 7.
SUSIN

PLAN 3.4
Suspend Subprogram Awaiting Flag-setting Interrupt or AUTO

Function
Suspend the current program, subprogram or member until or unless further action is required of it by an interrupt from a flag-setting peripheral device or by an AUTO operation.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SUSIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>2 (but see Note 3 below)</td>
</tr>
<tr>
<td>Operand</td>
<td>Blank</td>
</tr>
</tbody>
</table>

Execution

If one or more AUTO operations or flag-setting interrupts from any peripheral assigned to this program have caused the M or P indicators to be set, then SUSIN clears these indicators and the program proceeds to the next instruction.

If the M and P indicators are clear, then the current member is suspended until such time as either an AUTO operation referring to it or a flag-setting interrupt (from any peripheral assigned to this program) occurs. When reactivated by such an occurrence the program member commences from the instruction immediately following the SUSIN instruction.

Thus if a flag-setting peripheral is controlled by a subprogram and flag-setting interrupt occurs, then if the program member has been suspended by a SUSIN it will be reactivated, or if it is currently active the P indicator will be set and may be tested and subsequently cleared by a SUSIN.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
The SUSIN statement has an M-field (but N(M) must be zero).

Machine Code Listing
164 2 0 0

Notes
1 On being reactivated a program member should ascertain the reason for the reactivation.
2 When a program member is suspended, control returns to the next member of the highest priority at the point at which that member was last interrupted.
3 The following variants of the SUSIN instruction are available for use only on processors having the Priority Interrupt Feature. They may be used only in a program which includes a Priority Member (member 5). In such programs there is a third indicator, the E indicator, associated with each non-priority member. This E indicator is set for each non-priority member which is not under a SUSIN 3 suspension if the Priority Member issues a SUSIN 4 instruction. Details of these instructions follow:

(a) SUSIN 3

(i) In a member other than the Priority Member, the effect of SUSIN with $X = 3$ is similar to the effect with $X = 2$, except that the state of the E indicator is also considered.

The SUSIN 3 instruction tests the M, P and E indicators of the issuing member, and if any is set, clears all three and proceeds to the next instruction. If all are clear the member is suspended until an event which would otherwise set one of them occurs.

(ii) In the Priority Member, if there have been one or more interrupts from any Priority Device since the member was last activated, the SUSIN 3 instruction has no effect other than clearing C; the Priority Member proceeds to the next instruction. If there have been no interrupts from a Priority Device since the member was last activated, the member is suspended until either such an interrupt occurs or it is reactivated by an AUTO 5 0 in another member.
The machine code listing for SUSIN 3 is 164 3 0 0.

(b) SUSIN 4

This instruction is available only to the Priority Member itself. It operates for the Priority Member in the same way as SUSIN 3, but in addition, whether or not the Priority Member is suspended, it reactivates all members which are under a SUSIN 3 suspension, and sets the E indicator for all other members which are not under a SUSIN 3 suspension.

The machine code listing for SUSIN 4 is 164 4 0 0.

4 In a program not having a Priority Member, the effect of SUSIN with any value other than 1 or 2 in the accumulator field is undefined. SUSIN with 1 in the accumulator field is the equivalent of SUSAR.

In a program having a Priority Member, the effect of SUSIN in a non-priority member with a value other than 1, 2 or 3 in the accumulator field is undefined. In a Priority Member, the effect of SUSIN with a value other than 3 or 4 in the accumulator field is undefined.

5 If the SUSIN instruction with X = 2 is used on a processor that does not have subprogramming facilities but to which one or more flag-setting devices are attached, then the following subset of the above description applies:

If one or more flag-setting interrupts from any peripheral assigned to this program have caused the P indicator to be set since the last issued SUSIN instruction, then SUSIN clears the indicator and the program proceeds to the next instruction. If the P indicator is already clear, the SUSIN instruction suspends the program until a flag-setting interrupt occurs, at which time the program proceeds to the next instruction and the P indicator is left clear.
SUSMA

Test and Transfer to Common Storage

Function
Conditionally alter the contents of one word-pair in common storage (as between members of a program) and omit the instruction immediately following the SUSMA instruction.

Format
Operation Code: SUSMA
Accumulator: X
Operand: N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the location in common storage immediately preceding the location to be tested. If the contents of N + 1(M) are zero, then they are made non-zero and the contents of X are stored in N(M).

Execution
If a program consists of more than one member, this instruction is used to control their common storage (e.g. a buffer area) and to communicate, as between members, the current state of that storage. The instruction tests the common storage word N + 1(M) (where N(M) is the modified operand) and if it is equal to zero then the following action occurs:

1. The word N + 1(M) is made not zero.
2. The word N(M) is made equal to the contents of the accumulator X.
3. A branch is made to the instruction in the location next but one following the SUSMA instruction.

If, on testing, the word N + 1(M) is not equal to zero, then there is no action and the program proceeds to the instruction immediately following the SUSMA statement.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification
The SUSMA statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits are taken as the operand.

Example
A program member may, for instance, initiate a transfer into a buffer area BUFF + 2, and when it has completed the transfer it may set the word BUFF + 1 to zero. Another member may test the state of the transfer by the instruction sequence.

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>B</th>
<th>M</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDN</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUSMA</td>
<td>4</td>
<td>BUFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRN</td>
<td>ELSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORK</td>
<td>LDX</td>
<td>5</td>
<td>BUFF+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
If the transfer has been completed, BUFF + 1 will be equal to zero and the SUSMA will make it non-zero again and will put 1 (the contents of 4) into BUFF as an indication to another member that the current member is about to use the information just transferred. A branch is then made to WORK, to begin work on the data in the buffer. If the transfer had not been completed, then BUFF + 1 when tested would be non-zero and the program would proceed to the next instruction, this being a branch to the routine ELSE.

Machine Code Listing

162 X M N

Notes

1. By testing and altering common storage in one statement the possibility of another member corrupting the data is eliminated.

2. The operand of this statement must not equal zero. Indeterminate results will occur if it addresses storage that is not common.

3. If the SUSMA instruction is the operand instruction of an OBEY instruction, then the program member will proceed to the instruction located next or next but one after the OBEY instruction, not to that next or next but one after the SUSMA.
**SUSTY**

**PLAN 1.2.3.4**

**Function**
To suspend the program and type a variable-length message of up to 40 characters.

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SUSTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   a. A symbolic name referring to a lower data location.
   b. A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   a. A decimal integer in the range 0 to 4095.
   b. An octal integer in the range #0 to #7777.
   c. A previously defined absolute symbol with a value in the range 0 to 4095.

3. A literal (not PLAN 1).

The operand N(M) refers to a location which holds an index word consisting of a count of the number of characters (maximum 40) and the address of the first character of the message (which must be the first character, n0, in a word).

**Execution**
The SUSTY instruction suspends the program member and types a message on the console typewriter. Other members of the program may or may not be suspended. The issuing program member remains suspended awaiting operator action.

The message may be up to 40 characters long, and is referenced by means of an index word. It must not contain the characters represented by #74 to #77. The index word, held in the location specified by N(M), indicates the number of characters and the address of the first character in the message. The first character must be in the n0 position of a word.

The instruction can be modified. On dual and multiprogramming processors, each message is preceded by n\$name; HALTED: - (or an abbreviation thereof) where n is the program member number and name is the name of the program sending the message. On single-program processors, the message is preceded by HALTED only (or an abbreviation thereof).

C is not used and will be left clear.

V is not used and remains unchanged.

**Modification**
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

**Execution in Extended Data Mode**
When operating in extended data mode the operand N(M) may refer to either of the following:

1. A location which holds an index word as defined above.

2. A location which is the first word of a two word control area. The format of the control area is:

<table>
<thead>
<tr>
<th>Word N(M)</th>
<th>Bits 0 to 17</th>
<th>zero</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bits 18 to 23</td>
<td>character count</td>
</tr>
</tbody>
</table>
Word N+1(M) Bits 0 to 1 zero

Bits 2 to 23 address of the word containing the first character.

Whether the instruction is to use a one word or a two word control area is determined by whether or not bits 0 to 9 of word N(M) are zero.

Example

At a certain point in a program it is required to suspend the program and type out the message CHANGE TO INVOICE STATIONERY. This could be achieved by the following sequence:

| LABEL   | OPERATION | ACC. | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---------|-----------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| supply  |           |      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| message | 28:CHANGE TO INVOICE STATIONERY. |      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| count   | 28 / message - 0. |      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| program |           |      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|          |           |      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| susy    | count     |      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Machine Code Listing

160 0 M N

Notes

1 The compiler inserts 0 in the accumulator field of the machine-code instruction generated to distinguish it from DISTY and DELTY.

2 The use of this operation on a 1901 processor without a console typewriter is not recommended. Only the address of the index word will be displayed on the control panel lights, which will not be meaningful.

3 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

4 If the program is being run under the control of a GEORGE 1 or GEORGE 2 operating system, the operating system will intervene. Instead of HALTED, the console message will appear as DISPLAY.
**SUSWT**

**Function**  
Suspend the program and type the message HALTED, followed by a two-character code.

**Format**  
- **Operation Code**: SUSWT  
- **Accumulator**: Blank  
- **Operand**: This may be:  
  1. Up to four octal digits (modifiable).  
  2. Two symbols which can be represented in alpha shift preceded by 2H (unmodifiable); not PLAN 1.

**Execution**

The SUSWT instruction suspends the program member and causes a message to be typed on the console typewriter. Other members of the program may or may not be affected. The issuing program member remains suspended awaiting operator action.

The message includes the word HALTED (or an abbreviation thereof) followed by two characters from the card/printer set. The characters $, ], †, and = (#74 to #77) may not be used. The two characters are formed as follows:

1. If the operand was written as up to four octal digits, preceded by # and possibly modified, then the corresponding 12-bit quantity (if modified, only the least significant 12 bits are used) is treated as two 6-bit characters and the corresponding symbols are printed.
2. If the operand was written as two symbols preceded by 2H, then these symbols are printed. (Not PLAN 1.)

On dual and multiprogramming processors, each message is preceded by n#NAME, where n is the program member number and NAME is the name of the program sending the message.

- C is not used and will be left clear.
- V is not used and remains unchanged.

**Modification**  
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand. Bits above bit 12, which can only arise through modification, will be ignored in the output message. (See Execution.)

**Machine Code Listing**

161 0 M N  
where N is the decimal value of the operand.

**Notes**

1. The compiler inserts 0 in the accumulator field of the machine-code instruction generated to distinguish it from DISP and DEL.
2. On the 1901 processor without a console typewriter the 12-bit equivalent of the two characters is displayed on the control panel lights.
3. If the program is being run under the control of a GEORGE 1 or GEORGE 2 operating system, the operating system will intervene. Instead of HALTED, the console message will appear as DISPLAY.
TEST

Test Switch

Function       Test the state of bit N of word 30 (switch word).

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>X</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:
An absolute expression in the range 0 to 1023, e.g.
(a) A decimal integer in the range 0 to 1023.
(b) An octal integer in the range #0 to #1777.
(c) A previously defined symbol with a value in the range 0 to 1023.

The operand N(M) specifies the bit position to be tested and should have a value in the range 0 to 23.

Execution

TEST is a macro-instruction which results in the generation of three machine-code instructions. Its effect is to test bit N of word 30 and to indicate the state of that bit in X. If the bit is a 1 then the content of X, the specified accumulator, will be non-zero. If the bit is a 0 then the content of X will be equal to zero.

C is not used and will be left clear.

V is not used and remains unchanged.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example

It is required to test the state of bit 14 of the switch word (word 30). If the switch is set (i.e. = 1) branch to SET.

| LABEL | OPERATION | ACC | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  | OPERAND | REL. |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-------|------|
|       | TEST      | X   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |   30  |
|       | BRNZ      | SET |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |     |

The TEST macro-instruction above will generate the following series of machine-code instructions:

| LABEL | OPERATION | ACC | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  | OPERAND | REL. |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-------|------|
|       | LDCT      | #400|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |      |
|       | SRL       | X   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |      |
|       | ANDX      | 30  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |      |
|       |           |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |      |

Machine Code Listing

124 X 0 #400
112 X M N
020 X 0 30

Notes

1. The state of the bit can be found by testing the content of the specified accumulator (e.g. with a BNZ instruction).
FUNCTION
Test the contents of X and, if they are less than the contents of N(M), set C.

FORMAT
Operation Code: TXL
Accumulator: X
Operand: N(M)

N may be:
1. A relative expression which refers to a lower data location, e.g.,
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2. An absolute expression in the range 0 to 4095, e.g.,
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.
3. A literal (not PLAN 1).

The operand N(M) refers to the location whose contents are to be compared with the contents of X.

EXECUTION
The TXL instruction causes the contents of X to be compared with those of location N(M). According to the result of the comparison, the state of C is affected as follows:

- Contents of X less than contents of N(M) : C is set
- Contents of X equal to contents of N(M) : C is unchanged
- Contents of X greater than contents of N(M) : C is cleared.

Since the TXL instruction may be required for use with alphanumeric characters as well as with numbers, all words are considered to be 24-bit positive binary numbers and thus no significance is attributed to the sign bit. Therefore when comparing numbers, this instruction will only indicate the correct relationship between them when either both are positive or both are negative. Otherwise a negative number would always be considered greater than a positive number, as it has a 1-bit in the B0 position.

C is used and may be set, cleared or left unchanged (see above). It is not recommended that this instruction should be used with C set unless by a previous TXL instruction.

V is not used and remains unchanged.

MODIFICATION
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

EXAMPLE
It is required to test the contents of a storage location called DATA. If the contents of DATA are less than 75 the program must branch to an instruction labelled NEXT.
### Machine Code Listing

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>08</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>48</th>
<th>56</th>
<th>64</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX</td>
<td>5</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TXL</td>
<td>5</td>
<td>&quot;76&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCS</td>
<td>6</td>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROGRAM</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

### Notes

1. The state of the Carry register (C) may be tested by the conditional branch instructions BCC and BCS.

2. The comparison functions TXL and TXU are designed to simplify list searching and multiple-length comparisons.

3. In order that the TXL instruction can be used with alphanumeric data all words are considered as 24-bit positive binary numbers, i.e. no special significance is attached to B0. No problem arises when comparing numbers with the same sign but if the numbers being compared have opposite signs it is necessary first to invert the sign bits of both numbers before using the TXL instruction. The advantages of using a TXL instruction rather than a subtract instruction are:

   (a) The state of X is not altered.
   (b) There is no risk of setting V.
   (c) BCS and BCC branches are faster than BZE or BNZ.

4. If operand type 1 (a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

5. When using TXL instructions to do a multi-word comparison it is necessary to start the comparison at the least significant end.
PLAN 1,2,3

Function
Test the contents of the two locations X and N(M) for equality, and set C if the words are unequal; otherwise C will be left unchanged.

Format
Operation Code  TXU
Accumulator  X
Operand  N(M)

N may be:
1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.
2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.
3 A literal (not PLAN 1).
The operand N(M) refers to the location whose contents are to be compared with the contents of X.

Execution
The TXU instruction tests the contents of the two locations X and N(M) for equality, and sets C if they are not equal. If the words are equal, C is left unchanged. The fact that C is not unset by TXU permits the programmer to compare a complete multi-length quantity, before testing C.

C is used and may be set or left unchanged (see above). It is not recommended that this instruction should be used with C set unless by a previous TXU instruction.

V is not used and remains unchanged.

Modification: This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Example
It is required to test the contents of a storage location called DATA. If the contents of DATA are not equal to 75, the program must branch to an instruction labelled NEXT.

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D.N</td>
<td>1</td>
<td>15</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T.X.U</td>
<td>1</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.C.S</td>
<td>1</td>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Machine Code Listing
026 X M N
Notes

1 The state of the Carry register (C) may be tested by the conditional branch instructions BCC and BCS.

2 The comparison functions TXL and TXU are designed to simplify bit searching and multiple-length comparisons.

3 The advantages of using the TXU instruction rather than a subtract instruction are that
   (a) the state of X is not altered,
   (b) there is no risk of setting V,
   (c) BCS and BCC branches are faster than BZE or BNZ.

4 If operand type 1(a) is an undefined symbol the compiler will allocate a location in lower data to the symbol (not PLAN 1).

5 When using TXU instructions to do a multi-word comparison, the order of comparison is not significant.
PLAN 3.4
(Magnetic Tape Macro-instruction)

Function
Unload a magnetic tape.

Format
Operation Code UNL
Accumulator A decimal digit in the range 0 to 7 specifying the program's unit number for the magnetic tape unit.
Operand Blank

Execution
UNL causes the tape on the specified unit to be closed, rewound and released from the program. The specified magnetic tape unit is also put off-line to the computer system.

Modification The UNL macro-instruction has no M-field.

Machine Code Listing
157 X 0 n LT

when n is the relative address in the literals table of the first of two consecutive locations which hold a control area of the form:

first word: 5/#1007
second word: 0

Notes
1 This is one of a set of magnetic tape macro-instructions which permit the execution of simple basic functions on magnetic tape without requiring the programmer to set up control areas.
(See also BSP, BTM, CLOSE, FTM, REW, SCR, WTM.)
WTM

Write Tape Mark

**Function**  
Write a tape mark to a magnetic tape.

**Format**  
- **Operation Code**  
  WTM
- **Accumulator**  
  A decimal digit in the range 0 to 7 specifying the program's unit number for the magnetic tape unit.
- **Operand**  
  Blank

**Execution**  
WTM causes a tape mark to be written to the tape on the specified magnetic tape unit. It does *not* write a qualifier block; this must be inserted by a PERI operation or an MTH pseudo-operation.

**Modification**  
The WTM macro-instruction has no M-field.

**Machine Code Listing**

```
157  X  0  n  LT
```

when \( n \) is the relative address in the literals table of the first of two consecutive locations which hold a control area of the form:

- first word: \( 5/ \#5 \)
- second word: 0

**Notes**

1 This is one of a set of magnetic tape macro-instructions which permit the execution of simple basic functions on magnetic tape without requiring the programmer to set up control areas. (See also BSP, BTM, CLOSE, FTM, REW, SCR, UNL.)
PLAN 4 (M)PLT ONLY
(Macro-instruction)

Function
Load floating-point zero into the extended floating-point accumulator and clear N(M) to N + 3(M).

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>ZEFPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of four consecutive locations that are to be zeroized.

Execution
The ZEFPS instruction will generate two machine code instructions. The effect of the instruction is to load floating-point zero into the extended floating-point accumulator and store the contents of the floating-point accumulator in the double-length location N(M) to N + 1(M). The contents (floating-point zero) of the floating-point accumulator extension are then stored in the double-length location N + 2(M) and N + 3(M).

FOVR is left clear.
C is not used and will be left clear.
V will remain unchanged.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing
137 5 M N
137 2 M N

Notes
ZEFPS N(M) is equivalent to
ZFPS N(M)
SAE N(M)
ZFPS

Zeroize Floating-point Accumulator and Store

Function

Load floating-point zero into the extended floating-point accumulator and clear N(M) and N + 1(M).

Format

Operation Code
ZFPS

Accumulator
Blank

Operand
N(M)

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to the first of two consecutive locations that are to be zeroized.

Execution

Floating-point zero is loaded into the extended floating-point accumulator. The contents (floating-point zero) of the floating-point accumulator are then stored in the double-length location N(M) and N + 1(M).

FOVR is left clear.

C is not used and will be left clear.

V will remain unchanged.

Modification This statement has an M-field. When modified the least significant 15 bits of N + M are taken as the operand. In extended data mode, the least significant 22 bits of N + M are taken as the operand.

Machine Code Listing

137 5 M N
Chapter 5  Pseudo-operations

INTRODUCTION

This chapter deals with the macro-instructions relating to the Input/Output Generator, the Variable Limit Register, the Storage Device Housekeeping systems, the Overlay system and the Dump and Restart package. These facilities are only available with PLAN 3 and PLAN 4, except for the Cassette Tape Housekeeping system, which can be used with PLAN 1.

The sections on these items are not intended to be complete user specifications. Full information is provided in other ICL manuals, and the reader is referred to these in the appropriate sections below.

All the pseudo-operations described in this chapter are available in PLAN 4 with #XPLT.

THE INPUT/OUTPUT GENERATOR

The pseudo-operations associated with the Input/Output Generator enable the programmer to handle automatically all input, output, distribution and radix conversion (that is, binary to decimal and binary to sterling or vice versa) of defined records. The data referenced by these operations is usually associated with card readers and punches, paper tape readers and punches, and line printers, although data associated with magnetic tape files can also be referenced if there is a similar distribution.

The three Input/Output Generator pseudo-operation statements are:

INPUT
INDIS
OUT

When the PLAN 3 or PLAN 4 compiler encounters one of the Input/Output Generator pseudo-operations, the compiler recognizes it as such and generates object coding composed mainly of CALLS to appropriate subroutines, together with the required parameters. The Input/Output Generator subroutines form part of the Master Library Tape. Since only those subroutines required to deal with specified peripherals and record formats are incorporated into the object program, storage requirements vary according to the number of facilities used.

Descriptions of the three Input/Output Generator statements are given in the following pages, defined according to the format followed in Chapter 4. A full description of the Input/Output Generator is to be found in the ICL 1900 Series manual ‘Basic Peripherals’.

There are two implementations of the Input/Output Generator. One works only in programs which are compiled in compact mode (15AM and DBM); the other works in any mode (any logical combination of 15AM, 22AM, DBM and EBM) so long as the program runs in the same mode as that in which it was compiled. The specifications of INPUT, INDIS and OUT which follow apply to both versions, except that the sterling editing facilities have been removed in the mode compatible implementation.
INPUT

Input a Record

Function
Causes the next record of input to be presented.

Format
Operation Code       INPUT
Accumulator          Blank
Operand               A, Z, where A is the input peripheral name (e.g. CR3), and Z is the label that identifies the end-of-file routine.

Execution
The INPUT instruction is a macro-instruction that results in the incorporation in the object program of the appropriate routines for the presentation of the next type of input record.

Each INPUT statement referring to a particular peripheral unit must be followed (although not necessarily immediately in the coding sequence) by an INDIS statement for that unit.
PLAN 3

Function: Breaks records down into individual fields.

Format:
- Operation Code: INDIS
- Accumulator: Blank
- Operand: A, Z, where A is the input peripheral name (e.g. TR2) or the name of a user buffer area (which must not be the same as a peripheral name) for records read from magnetic tape; Z is the label that identifies the user's error routine.

Execution:
The INDIS instruction is a macro-instruction that results in the incorporation in the object program of the appropriate routines for breaking input records down into their individual fields. These fields are specified in distribution lines following the INDIS statement, their types being defined by a single letter in the operation field, as follows:

- A: Alphanumeric (to be held in character form)
- B: Blank (or Ignore)
- N: Numeric (to be converted to binary)
- S: Sterling (to be converted to binary)

The distribution lines are always followed by END in the operation field. The entry in the operand field of each distribution line depends on the type of input, which may be fixed alphanumeric, variable alphanumeric, fixed numeric (integers only), fixed numeric (integers and fractions), variable numeric (integers only), variable numeric (fractions only), variable numeric (integers and fractions), fixed sterling (integers only), fixed sterling (fractional pence), variable sterling (integers only), variable sterling (fractional pence), or a field to be ignored. The permitted formats of these options are specified below.

The following abbreviations are used in the format definitions of types of distribution lines which can be used following an INDIS statement.

- * = the field terminator which may be any character except 0 to 9, V, /, L, S, D, &,, V, =, +, or .
  It may also be two characters the first of which is + (#76).
- a = the number of words of store required to hold the result of the conversion. The number must equal one or two.
- b = the number of bits of the above word or words which are to be reserved for the fractional part of the field which is converted. In the format definitions b is optional and may therefore be omitted, together with the point preceding it.
- p = the number of characters before the decimal point.
- q = the number of characters after the decimal point.
- L = the letter which in a sterling distribution line indicates that the immediately preceding character(s) refer to the pounds field.
- S = the letter which in a sterling distribution line indicates that the immediately preceding character(s) refer to the shillings field.
- D = the letter which in a sterling distribution line indicates that the immediately preceding character refers to the pence field.

The option of 2, &,, or : followed by D indicates respectively:
- Two pence characters (2D).
One pence character where tenpence is represented by & and elevenpence by - (&D).

One pence character where tenpence is represented by : and elevenpence by ; (:D).
The option of . or V indicates the presence (.) or absence (V) in the data of a punched decimal point.

\[ n = \text{the address modifier and is always optional.} \]
\[ n = \text{the symbolic name of an area of store which has been previously defined under a #LOWER or #UPPER directive.} \]

In the case of numeric and sterling fields to be input and converted to binary:

\[ n(m) = \text{the address of the first word of a one or two word store area into which the result of the} \]
\[ \text{conversion is to be placed. This must be a word address.} \]

In the 'pictures' that follow, if a capital letter is present (e.g. L), then this letter must be used when
the picture is written on the coding sheet. If a small letter is present (e.g. p), then the appropriate
numerical value must be substituted when the picture is written on the coding sheet.

In the case of \( n \), the appropriate symbolic name will usually be substituted rather than a number.

**Fixed Alphanumeric Input**  \( A \quad e/n, j(m), s \)
where \( e \) = the number of characters to be input,
\[ n, j(m) = \text{the optionally modified character address to which the first character is to be} \]
\[ \text{transferred,} \]
\[ s = \text{the number of characters to be transferred, that is,} \quad e + \text{a number of spaces (which} \]
\[ \text{may be zero).} \]

**Notes**
1. \( e \) must be a positive non-zero number < 129.
2. \( s \) need not be specified in which case it will be assumed = \( e \).
3. \( 0 < s - e < 129 \).

**Variable Alphanumeric Input**  \( A \quad */n, j(m), s \)
where \( n, j(m) = \text{the optionally modified character address to which the first character is to be} \)
\[ \text{transferred,} \]
\[ s = \text{the maximum number of characters allowed in the user-defined store area.} \]

**Notes**
1. \( s \) must be present.
2. If the number of characters transferred is less than \( s \) the remainder of the user-defined store area
will be space-filled.

**Fixed Numeric Input (Integers only)**  \( N \quad e/n(m), a, b \)
where \( e \) = the number of decimal characters to be input and converted to binary.

**Notes**
1. The maximum value for \( e = 19 \).
2. In this case \( b \) and the point preceding it will normally be omitted.
Fixed Numeric Input (Integer and Fraction) \[ N \quad p \left\{ \frac{V}{\cdot} \right\} q/n(m), a, b \]

**Notes**
1. The maximum value of \( q = 14 \).
2. \( q \) may not be zero.
3. \( p \) may equal zero.

Variable Numeric Input (Integers only) \[ N \quad */n(m), a, b \]

**Notes**
1. In this case \( b \) and the point preceding it will normally be omitted.

Variable Numeric Input (Fractions only) \[ N \quad 0 \left\{ \frac{V}{\cdot} \right\} */n(m), a, b \]

Variable Numeric Input (Mixed Integer and Fraction)
(a) Length of fractional part fixed. \[ N \quad *\left\{ \frac{V}{\cdot} \right\} q/n(m), a, b \]
(b) Length of integral part fixed. \[ N \quad p\left\{ \frac{V}{\cdot} \right\} */n(m), a, b \]
(c) Length of both integral and fractional parts variable. \[ N \quad **/n(m), a, b \]

**Notes**
1. The maximum value of \( q = 14 \).
2. \( q \) may not be zero.
3. The use of the double terminator in the distribution format is a convention. In the actual data, if the field is a mixed number, a decimal point must be punched to act as a terminator for the integral part of the field, the terminator shown in the format terminating the fractional part of the field. If the field is entirely integral it is only necessary to punch the terminator shown in the format.

Fixed Sterling Input (Integers only)
(a) Pounds, shillings and pence. \[ S \quad xL2S \left\{ \frac{2}{\cdot} \right\} D/n(m), a, b \]
(b) Pounds and shillings. \[ S \quad xL2S/n(m), a, b \]
   where \( x = \) the number of digits in the pounds field which must be greater than zero and less than 20,
   \[ 2S = \] the shillings field (fixed format), i.e. there must be two characters to represent shillings.
(c) Shillings and pence. \[ S \quad xS \left\{ \frac{2}{\cdot} \right\} D/n(m), a, b \]
(d) Shillings. \[ S \quad xS/n(m), a, b \]
   where \( x = \) the number of digits in the shillings field, and must be greater than zero and less than 20.
Notes
1 In the above cases, \( b \) and the point preceding it will normally be omitted.

Fixed Sterling Input (Fractional Pence)

(a) Pounds, shillings and pence. \( S \ xL2S \{ \ 2 \ \} \ D \{ \ V \ \} q/n(m), a, b \)

where \( x = \) the number of digits in the pounds field,
\( 2S = \) the shillings field (fixed format), i.e. there must be two characters to represent shillings.

(b) Shillings and pence. \( S \ \times S \{ \ 2 \ \} \ D \{ \ V \ \} q/n(m), a, b \)

where \( x = \) the number of digits in the shillings field and must be greater than zero and less than 20.

Notes
1 The number of digits in the pounds field must be greater than zero and less than 20.
2 \( q \) may not be zero nor exceed 14.

Variable Sterling (Integers only)

(a) Pounds, shillings and pence. \( S \ \times L2S \{ \ 2 \ \} \ D/n(m), a, b \)

(b) Pounds and shillings. \( S \ \times L2S/n(m), a, b \)

where \( 2S = \) the shillings field (fixed format), i.e. there must be two characters to represent shillings.

(c) Shillings and pence. \( S \ \times S \{ \ 2 \ \} \ D/n(m), a, b \)

(d) Shillings. \( S \ \times S/n(m), a, b \)

Notes
1 The number of digits in the pounds field must be greater than zero and less than 20.
2 The number of digits in the shillings field must be greater than zero and less than 20.
3 In the above cases \( b \) will normally be omitted together with the point preceding it.

Variable Sterling Input (Fractional Pence)

(a) Integral part known; pounds, shillings and pence. \( S \ xL2S \{ \ 2 \ \} \ D \{ \ V \ \} */n(m), a, b \)

where \( x = \) the number of digits in the pounds field,
\( 2S = \) the shillings field (fixed format), i.e. there must be two characters to represent shillings.

Notes
1 The number of digits in the pounds field must be greater than zero and less than 20.
(b) Integral part known; shillings and pence.  \[ S \ xS^{2} \ \& \ D \ V^{2} \ n(m) \ \& \ x \ a \ b \]

where \( x \) = the number of digits in the shillings field, and must be greater than zero and less than 20.

(c) Fractional part known; pounds, shillings and pence.  \[ S ^{L2S} \ x^{V} \ \& \ D \ n(m) \ \& \ a \ b \]

where \( 2S \) = the shillings field (fixed format), i.e. there must be two characters to represent shillings.

Notes
1. The number of digits in the pounds field must be greater than zero and less than 20.
2. \( q \) must not be zero nor exceed 14.

(d) Fractional part known; shillings and pence.  \[ S ^{S} \ x^{V} \ \& \ D \ n(m) \ \& \ a \ b \]

Notes
1. The number of digits in the shillings field must be greater than zero and less than 20.
2. \( q \) must not be zero nor exceed 14.

(e) Length of both integral and fractional parts variable; pounds, shillings and pence.

\[ S ^{L2S} \ x^{V} \ \& \ D \ n(m) \ \& \ a \ b \]

where \( 2S \) = the shillings field (fixed format), i.e. there must be two characters to represent shillings.

Notes
1. The number of digits in the pounds field must be greater than zero and less than 20.
2. The use of the double terminator in the distribution format is a convention. In the actual data, if the field is a mixed number, a decimal point must be punched to act as a terminator for the integral part of the field; the terminator shown in the format terminates the fractional part of the field. If the field is entirely integral it is only necessary to punch the terminator shown in the format.

(f) Length of both integral and fractional parts variable; shillings and pence.

\[ S ^{S} \ x^{V} \ \& \ D \ n(m) \ \& \ a \ b \]

Notes
1. The use of the double terminator in the distribution format is a convention. In the actual data, if the field is a mixed number, a decimal point must be punched to act as a terminator for the integral part of the field; the terminator shown in the format terminates the fractional part of the field. If the field is entirely integral it is only necessary to punch the terminator shown in the format.

**Input to be Ignored**  \[ B \ x \]

where \( x \) = the number of characters to be skipped.
Notes

1 \[ 0 < x < 129 \]

Example

Input records read in from paper tape and distributed consist of:

A designation character that is ignored.

A variable alphanumeric field of up to thirty characters terminated by *.

A numeric field of four digits to be held in one word.

A numeric field of two digits to be held in one word.

A variable numeric field, terminated by a newline character, to be held in two words.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>6</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDIS.</td>
<td>TRQ,ERROR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>NAME,30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>QTY,1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>DISCT,1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>NEBULL,2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chapter 5
OUT

**PLAN 3**

**Distribute and Output Fields**

**Function**

Distributes named fields to a user buffer area or distributes and outputs them to a peripheral. Only fixed length output is possible.

**Format**

- **Operation Code**: OUT
- **Accumulator**: Blank
- **Operand**: A, Z, where A is the output peripheral name (e.g. LP0) or the name of a user buffer area (which must not be the same as a peripheral name); Z is the label that identifies the user's error routine.

**Execution**

The OUT instruction is a macro-instruction that results in the incorporation in the object program of the appropriate routines for distributing fields to a user buffer area or distributing and outputting them to a peripheral. These fields are specified in distribution lines following the OUT statement, their types being defined by a single letter in the operation field, as follows:

- A Alphanumeric (held in character form)
- B Blank
- N Numeric (to be converted from binary)
- S Sterling (to be converted from binary)
- H Header (for headings)

For output to a line printer, an additional statement immediately follows the OUT statement and takes the form:

- **Operation Field**: L
- **Accumulator**: Blank
- **Operand**: \#xy, where \#xy is the print/paper feed control character.

For output to a paper tape punch, an additional statement may optionally follow the field descriptions; it takes the form:

- **Operation Field**: P
- **Accumulator**: Blank
- **Operand**: n, where n is 1, 2 or 3, representing a newline, line feed or form feed character respectively.

The distribution lines are always followed by END in the operation field.

*The following abbreviations are used in the formal definitions of types of distribution lines, which can be used following an OUT statement.*

- **a** = the number of words of store in which the binary number to be converted is held. The number must equal one or two.
- **b** = the number of bits of the above word or words which contain the fractional part of the number to be converted. In the format definitions b is optional and may therefore be omitted together with the point preceding it.
- **p** = the number of characters to be output before the decimal point.
- **q** = the number of characters to be output after the decimal point.
- **L** = the letter which in a sterling distribution line indicates that the immediately preceding character(s) refer to the pounds field.
- **S** = the letter which in a sterling distribution line indicates that the immediately preceding character(s) refer to the shillings field.
\( D \) = the letter which in a sterling distribution line indicates that the immediately preceding character refers to the pence field.

The option of 2, &, or : followed by D indicates respectively:

- Two pence characters to be output (2D).
- One pence character to be output where tenpence is represented by & and elevenpence by - (\&D).
- One pence character to be output where tenpence is represented by : and elevenpence by ; (:D).

The option of . or V indicates that a decimal point is to be punched out (.) or alternatively the decimal point is assumed and will not be punched out (V).

\( m \) = the address modifier and is always optional.

\( n \) = the symbolic name of an area of store which has been previously defined under a \#LOWER or \#UPPER directive.

In the case of numeric and sterling to be converted from binary and output,

\( n(m) \) = the address of the first word of a one or two word store area which holds the binary number to be converted. This must be a word address.

\( d \) = a number which specifies the number of decimal places required with decimal currency editing. When present, it is enclosed in parentheses.

The following letters are used to denote editing facilities.

- \( Z \) = zero suppress.
- \( C \) = cheque protect with an asterisk.
- \( B \) = blank when entire field is zero.
- \( F \) = floating preceding minus sign.
- \( O \) = sign with overpunch on most significant character.
- \( S \) = pre-signing of negative field with up to three specified characters.
- \( A \) = post-signing of negative field with up to three specified characters.
- \( E \) = floating E sign.
- \( D \) = decimal currency.

Any picture containing more than one member of each group of editing facilities

1. Z or C
2. B
3. F or O or S or A
4. E
5. D

will be signalled as an error and the first facility specified will be accepted for compilation.

**NOTE:** If an error is made in the specification of editing facilities, this error will be flagged by the compiler. An error will result at run time only if none of the group F or S or A or O is mentioned, and the number is negative and there is a significant character in the first character position.

*In the pictures which follow, if a capital letter is present (e.g. L), then this letter must be used when the picture is written on the coding sheet. If a small letter is present (e.g. P), then the appropriate numerical value must be substituted when the picture is written on the coding sheet.*

In the case of \( n \) the appropriate symbolic name will usually be substituted rather than a number.

**Alpha Output**

\[ A / e / n, j(m), s \]

where

- \( e \) = the number of characters to be output, that is, \( s + a \) number of spaces (which may be zero),
- \( n, j(m) \) = the optionally modified character address from which the first character is to be transferred,
- \( s \) = the number of characters to be transferred from the user-defined store area.
Notes
1  e must be greater than zero and less than 129.
2  s need not be specified, in which case it is assumed equal to e.
3  0 < e-s < 129. If e is greater than s, then (e-s) space characters will be output.

Numeric Output (Integers only)  \[ N \{ e/n(m), a, b, \{ C \} \{ Z \} \{ F \} \{ O \} \{ S \} \{ A \} \} \{ D \} \]

where e = the number of numeric characters to be output.

Notes
1  e must be greater than zero and less than 129.
2  In this case b will normally be omitted together with the point preceding it.
3  a must be present.
4  An error is flagged if none of the group F or S or A or O is mentioned.
5  If F or O is used then Z must also be used; otherwise an error will be flagged.
6  If O is used then neither Z nor C may be used.
7  (d) may only be used with D editing. It is then optional; if it is omitted, two places of decimals will be given. d must be in the range 1 to 9 and must not be greater than e.

Numeric Output (Fractions only)  \[ N \{ q/n(m), a, b, \{ A \} \} \{ S \} \]

Notes
1  a, b must be present.
2  One of S or A must be specified. An error will occur at run time if neither S nor A is specified and the fractional field is negative.
3  None of Z, C, F or O can be used with this picture.
4  The value of q must be greater than zero and less than 15.

Numeric Output (Mixed Integer and Fractions)  \[ N \{ p/n(m), a, b, \{ Z \} \{ C \} \{ F \} \{ O \} \{ S \} \{ A \} \}

Notes
1  p \{ V \} q must be greater than zero and less than 129.
2  a, b must be present.
3  An error is flagged if none of the group F or S or A or O is mentioned.
4  If F is used then Z must also be used, otherwise an error will be flagged.
5  If O is used, then neither Z nor C may be used.
6  The value of q must be greater than zero and less than 15.
Sterling Output (Integers only)

(a) Pounds, shillings and pence. \( S \times L \times B \times S / n(m), a, b, \{ Z \} C \{ A \} B \{ F \} O \{ S \} A \)

(b) Pounds and shillings. \( S \times L \times B / n(m), a, b, \{ Z \} C \{ F \} O \{ S \} A \)

where \( xL \) = the number of digits in the pounds field which must be greater than zero and less than 129,
\( 2S \) = the shillings field (fixed format), i.e. there must be two characters to represent shillings,
\( xB \) = insertion of \( x \) blanks, i.e. space characters between the pounds and shillings and between the shillings and pence if these are pence.

(c) Shillings and pence. \( S \times S \times B \{ Z \} C \{ F \} O \{ S \} A \)

(d) Shillings. \( S \times S / n(m), a, b, \{ Z \} C \{ F \} O \{ S \} A \)

where \( xS \) = the number of digits in the shillings field and must be greater than zero and less than 129,
\( xB \) = insertion of \( x \) blanks, i.e. space characters between the shillings and pence.

Notes
1. \( a \) must be present.
2. In the above cases \( b \) and the point preceding it will normally be omitted.
3. Blanks may not be inserted before the pounds or after the pence fields.
4. An error is flagged if none of the group \( F \) or \( S \) or \( A \) or \( O \) is mentioned.
5. If \( F \) or \( £ \) is used then \( Z \) must also be used; otherwise an error will be flagged.
6. If \( O \) is used then neither \( Z \) nor \( C \) may be used.

Sterling Output (Fractional Pence)

(a) Pounds, shillings and pence. \( S \times L \times B \times S / n(m), a, b, \{ V \} C \{ F \} O \{ S \} A \)

where \( xL \) = the number of digits in the pounds field, which must be greater than zero and less than 129,
\( 2S \) = the shillings field (fixed format), i.e. there must be two characters to represent shillings,
\( xB \) = insertion of \( x \) blanks, i.e. space characters between the pounds and shillings and between the shillings and pence if these are pence.

(b) Shillings and pence. \( S \times S / n(m), a, b, \{ V \} C \{ F \} O \{ S \} A \)

where \( xS \) = the number of digits in the shillings field and must be greater than zero and less than 129,
\( xB \) = insertion of \( x \) blanks, i.e. space characters between the shillings and pence.

Notes
1. \( a, b \) must be present.
2. Blanks may not be inserted before the pounds or after the pence.
3. The value of \( q \) must be greater than zero and less than 15.
4. An error is flagged if none of the group F or S or A or O is mentioned.
5. If F or L is used then Z must also be used; otherwise an error is flagged.
6. If O is used, then neither Z nor C may be used.

Example

| LABEL | OPERATION | ACC. 12 13 14 15 16 17 18 19 20 31 32 33 34 35 36 37 38 39 40 44 45 46 47 48 49 50 51 52 53 54 55 56 | OPERAND |
|-------|-----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
|       | OUT       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
|       | L         |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
|       | S         |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
|       | END       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |

4322(7.72)  Chapter 5  13
THE VARIABLE LIMIT REGISTER

There are three pseudo operations available to enable users to handle the variable limit register. These pseudo operations are:

UNOFF
UNONC
UNONL
PLAN 4  (#XPLT ONLY)  

Function  
Switch off the limit on floating point underflow.

Format  

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>UNOFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>Blank</td>
</tr>
</tbody>
</table>

Execution  
The instruction sets up a location, \( N + 1 \), containing zero. \( N + 1 \) is used to clear the variable floating-point underflow limit switch. Therefore, floating-point underflow will occur when the exponent has a value of less than \(-256\).

Machine Code Listing  

130 1 N
UNONC

Switch a Limit on Floating Point Underflow and Clear Floating Point Accumulator

Function
A limit is set on floating-point underflow. When the limit is reached and underflow occurs the contents of the floating-point accumulator are cleared.

Format
- **Operation Code**: UNONC
- **Accumulator**: Blank
- **Operand**: K

where K is an absolute expression, lies in the range 0 to 248 and is a multiple of 8.

Execution
The instruction sets up a location, N + 1, containing (K \(\cap\) 248) + 1. N + 1 is used to set the limit on floating-point underflow. Therefore, floating-point underflow will occur when the exponent has a value K - 256. When floating-point underflow does occur, the contents of the floating-point accumulator will be left clear.

Machine Code Listing

```
130  1  N
```
PLAN 4 ( #XPLT ONLY )

Function
A limit is set on floating-point underflow. When the limit is reached and underflow occurs the contents of the floating-point accumulator are not cleared.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>UNONL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>K</td>
</tr>
</tbody>
</table>

where K is an absolute expression, lies in the range 0 to 248 and is a multiple of 8.

Execution
The instruction sets up a location, N + 1, containing \((K \cap 248) + 3\). \(N + 1\) is used to set the limit on floating-point underflow. Therefore, floating-point underflow will occur when the exponent has a value \(K - 256\). When floating-point underflow does occur, the contents of the floating-point accumulator will remain unchanged.

Machine Code Listing
130 I N

Notes
1 Where the value of K is zero, floating-point underflow will occur when the exponent has a value of less than \(-256\). However, the contents of the floating-point accumulator will not be cleared.
STORAGE DEVICE HOUSEKEEPING

ICL provide a wide range of devices which may be used as backing store for a computer. Input/output programming for storage devices is a complex procedure, and ICL provide software to reduce user programming effort. For software purposes there are at present three types of storage device: cassette tape, magnetic tape and the direct access devices. There are three housekeeping systems, the Cassette Tape Housekeeping system, the Magnetic Tape Housekeeping system and the Direct Access Housekeeping system. The routines that comprise these three systems are called by user programs (which must be in PLAN 3 or PLAN 4 to utilize the Direct Access and Magnetic Tape Housekeeping systems, or may be in PLAN 1 in the case of cassette tape) by means of storage device macro-instructions (SD macros).

The storage device to which any SD macro refers at compilation time is determined by the #CMODE directive (see Chapter 6, page 3). This directive associates the files to which SD macros refer with a specific storage device. One program can refer to different devices using SD macros so long as any file number is unique to one device. The storage device configuration for which a program has been compiled may be altered by changing the #CMODE directive and recompiling. To facilitate this process the directive is usually placed in a steering segment. The facility of changing the storage device configuration of a program was provided to enable storage device compatible programs to be written. When working in 'compatible mode', SD macros must contain the maximum information required. This means, for instance, that SDDEF must be defined as for direct access, the additional parameters being ignored if, in fact, magnetic tape is used. The #CMODE directive is not available in PLAN 1; the PLAN 1 compiler XPLQ generates its own version of the Cassette Tape Housekeeping macro-instructions in response to SD macros.

In PLAN 4 it is possible to call a choice of two versions of the Magnetic Tape Housekeeping or Direct Access Housekeeping systems: a compact mode version, or mode-compatible version that will operate in either 15-bit address mode or 22-bit address mode, DBM or EBM. The method of specifying the version required is described under the #HMODE directive in Chapter 6.

The SD macros are described in the following pages. At compilation time, calls are generated to the housekeeping routines appropriate to direct access devices, magnetic tape or cassette tape, depending on the information given in the #CMODE directive (except in the case of PLAN 1, as mentioned above). The particular routines called are named in accordance with the following conventions:

Housekeeping System

Direct Access (compact mode version).
Direct Access (mode-compatible version).
Magnetic Tape (compact mode version).
Magnetic Tape (mode-compatible version).
Cassette Tape, PLAN 3 or PLAN 4.
  For machines with a console typewriter.
  For machines without a console typewriter.
Cassette Tape, PLAN 1.
  For 4K machines without a console typewriter.

First three letters of routine's name

HRA
HDA
HMT
HTH
HDT
HDH
HDS

In general, the following letters of the routine's name are the same as the third and following letters of the SD macro that generates the call; but SDWRU and SDWRU in magnetic tape mode or cassette tape mode both generate calls to routines whose names end with the letters WR.

Certain SD macros generate calls in some housekeeping systems but not in others. This is either because the meaningful equivalent of the instruction is 'do nothing' (for example, SDWRU on direct access) or because the instruction has no meaning on the device specified (for example SDRRB on direct access).
To write a successful storage device compatible program it would be necessary to understand some of the hardware differences between the devices under consideration. This information may be found in the manuals 'Magnetic Tape', 'Cassette Tape' and 'Direct Access'.

All the housekeeping systems provide the user with two levels of file control, known as physical processing and logical processing. At the physical level, housekeeping transfers a unit of data (either a magnetic or cassette tape block or a direct access bucket) to or from the storage device, as instructed by the user. The processing of the contents of the block or bucket is done by the user. At the logical level housekeeping not only transfers data to and from the storage device but also processes the data (typically presenting a specified record to the user or accepting a record from him). Thus at the logical level the user program is reading and writing records without being concerned with peripheral transfers.

Physical level SD macros are:

- **SDRDB**
- **SDWRRB**
- **SDSUS**
- **SDRRB**

Logical level SD macros are:

- **SDCLB**
- **SDDEL**
- **SDIND**
- **SDRD**
- **SDRDP**
- **SDWR**
- **SDWRI**
- **SDWRRU**

Other general purpose SD macros are:

- **SDDEF** which must be used initially to define a file,
- **SDEND** which are used for closing files,
- **SDBTS** which are used with magnetic tape user sentinels,
- **SDFTS**
- **SDWRS**
- **SDWSS**
- **SDCLS**
- **SDFSS**
- **SDFES**
- **SDBSS**
- **SDLAB** which are used only with magnetic tape,*
- **SDBUF**
- **SDEXF** which is used only with direct access devices.
- **SDFAB** which is used only with direct access devices conforming to unified direct access standards.

Further information on handling files of data using the Magnetic and Cassette Tape Housekeeping systems is to be found in the 'Magnetic Tape' and 'Cassette Tape' manuals respectively. The Direct Access Housekeeping system is dealt with in the manual 'Direct Access'. A description of each of the SD macros is given in the following pages, in alphabetical order.

It must be strongly emphasized that the description of the function of many of these macro-instructions can only be meaningful in the context of wider knowledge. Moreover the function of any SD macro can alter as the result of exception conditions or such conditions as overflow in the case of the direct access devices. The publications mentioned above should help users to write programs successfully, incorporating housekeeping routines.

* These two groups of SD macros are not accepted by the compilers #XPLR and #XPLS.
SDBSS

Skip Back to Start-of-Subfile Sentinel

Cassette Tape and Direct Access

This SD macro-instruction refers only to magnetic tape, and causes no action on cassette tape and direct access devices.

Magnetic Tape

Function

Moves a magnetic tape backwards until the specified start-of-subfile sentinel is found, and leaves the tape positioned so that the read head is on the header label side of the sentinel.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDBSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) refers to a location which is the first word of the 20 word Subfile Definition Area.

Modification

This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Notes

1. The SDBSS macro-instruction may be used only for processing composite files.
2. The subfile specified in the Subfile Definition Area may be a simple or a composite subfile and may be at any level.
3. The SDBSS macro-instruction is not accepted by the compilers #XPLR and #XPLS.
4. The SDBSS macro-instruction should be used only where the object program is to be run on a system which has the skip back past tape mark facility.
SDBTS

PLAN 3.4

Cassette Tape and Direct Access
This SD macro-instruction refers only to magnetic tape, and causes no action on cassette tape and direct access devices.

Magnetic Tape
Function
Moves a magnetic tape backwards, leaving it positioned with the read head on the header label side of the first user sentinel encountered after a specified number of intervening user sentinels has been ignored.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDBTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N must be a decimal integer in the range 0 to 4095.

The operand N(M) specifies the number of user sentinels to be ignored.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Notes
1. If N(M) is zero or is omitted, the tape is moved backwards until a user sentinel is encountered, and is left positioned with the read head on the header label side of that sentinel.
2. The SDBTS macro-instruction may be used for processing a simple file or for processing a simple subfile within a composite file.
3. The SDBTS macro-instruction should be used only where the object program is to be run on a system which has the skip back past tape mark facility.
Set Up User's Buffers

Cassette Tape and Direct Access

This SD macro-instruction refers only to magnetic tape, and causes no action on cassette tape and direct access devices.

**Magnetic Tape**

<table>
<thead>
<tr>
<th>Function</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Operation Code</td>
<td>SDBUF</td>
</tr>
<tr>
<td></td>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td></td>
<td>Operand</td>
<td>N(M), L</td>
</tr>
</tbody>
</table>

N(M) may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 8 to 4095, e.g.
   - (a) A decimal integer in the range 8 to 4095.
   - (b) An octal integer in the range #10 to #7777.
   - (c) A previously defined absolute symbol in the range 8 to 4095.

N(M) refers to a location which contains the start address of the first buffer. If double buffering is specified, the location N + 1(M) must contain the start address of the second buffer. N(M) must not refer to an accumulator.

L may be:

1. A relative expression which refers to a lower data location, e.g.
   - (a) A symbolic name referring to a lower data location.
   - (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 7, e.g.
   - (a) A decimal integer in the range 0 to 7.
   - (b) An octal integer in the range #0 to #7.
   - (c) A previously defined absolute symbol with a value in the range 0 to 7.

3. A literal.

L refers to a location which contains in bits 0 to 8 the number of buffers required (1 or 2) and in bits 9 to 23 the block size.

**Modification** This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the first term of the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the first term of the operand.

**Notes**

1. The SDBUF macro-instruction may be used for processing simple or composite files.

2. The SDBUF macro-instruction should be the first instruction other than SDDEF to address the file, if user's buffers are required; except that if the magnetic tape is to be relabelled, the SDLAB macro-instruction may either precede or follow the SDBUF macro-instruction.
3. The size of each buffer area should be the block size plus four words.
4. If an SDBUF macro-instruction is used, the SDDEF macro-instruction that addresses the same file should have its N field set to zero; that is, no Housekeeping buffers should be specified.
5. The SDBUF macro-instruction is not accepted by the compilers #XPLR and #XPLS.
Close Batch/Bucket Early

Magnetic Tape, Cassette Tape and Direct Access

Function: Writes the contents of the housekeeping buffer areas to the storage device.

Format:
- **Operation Code**: SDCLB
- **Accumulator**: File number. This must be a decimal integer in the range 0 to 15.
- **Operand**: Blank.
SDCLS

PLAN 3, 4

Cassette Tape and Direct Access

This SD macro-instruction refers only to magnetic tape, and causes no action on cassette tape and direct access devices.

Magnetic Tape

Function
Clears a magnetic tape file's output buffers and writes end-of-subfile sentinel(s) for a specified subfile.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDCLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   a. A symbolic name referring to a lower data location.
   b. A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   a. A decimal integer in the range 0 to 4095.
   b. An octal integer in the range #0 to #7777.
   c. A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) refers to a location which is the first word of the 20 word Subfile Definition Area.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Notes

1. The SDCLS macro-instruction may be used only for processing composite files.

2. The subfile specified in the Subfile Definition Area may be a simple or a composite subfile and may be at any level.

3. If the specified subfile is a composite subfile, up to three levels of subfile may be closed by a single SDCLS macro-instruction, as follows (the buffers being cleared in each case):
   a. If the specified subfile is the lowest level of subfile open, the end-of-subfile sentinel is written for the specified subfile.
   b. If there is a subfile (whether simple or composite) open at one lower level, the end-of-subfile sentinels are written for the lower level subfile and for the specified subfile.
   c. If there are subfiles open at two lower levels, and the lowest level subfile open is a simple subfile, the end-of-subfile sentinels are written for the simple subfile, for the composite subfile at the next level up, and for the specified subfile.

Incorrect results are given if the specified subfile is a composite subfile and more than one lower level composite subfile is open.

4. If the subfile name and generation number are not required in an end-of-subfile sentinel then words 2 to 4 and word 6 of the appropriate Subfile Definition Area must be zeroised before an instruction is given which would close that subfile.

5. The SDCLS macro-instruction is not accepted by the compilers #XPLR and #XPLS.
SDCRE

Close Reel/Cassette Early

Direct Access
This SD macro-instruction refers only to magnetic tape and cassette tape, and causes no action on direct access devices.

Magnetic Tape and Cassette Tape

Function
Closes the current reel or cassette before either end-of-file or end-of-storage has been reached.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDCRE</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Accumulator</td>
<td>Blank.</td>
</tr>
<tr>
<td>Operand</td>
<td></td>
</tr>
</tbody>
</table>
Magnetic Tape and Cassette Tape

Function
Supplies information to the housekeeping system, enabling the file which is defined to be opened.

Format
Operation Code  SDDEF
Accumulator  File number. This must be a decimal integer in the range 0 to 15.
Operand  N, L, FDA, LABEL, MODE

N  =  The number of buffer areas to be allocated. For a file which is to be processed at the logical level this must be one or two, unless user's buffers are set up by an SDBUF macro-instruction, in which case N should be zero. For a file which is only to be processed at the physical level this should be zero.

L  =  The length, in words, of each buffer area. If N is one or two, L must be equal to the maximum block size of the file. If N is zero L should be zero.

FDA  =  The start address of the File Definition Area.

LABEL  =  The instruction in the user program to which control will be transferred should an exception condition arise. The different types of errors and exceptions are distinguished by codes put in words 10 and 11 of the program.

MODE  =  The mode of opening the file. This will be represented by a number in the range 1 to 9. See Note below for the meaning of these numbers.

Direct Access

Function
Supplies information to the housekeeping system, enabling the file which is defined to be opened.

Format
Operation Code  SDDEF
Accumulator  File number. This must be a decimal integer in the range 0 to 15.
Operand  N, L, FDA, LABEL, MODE, PROC, O, L1, LABEL1

N  =  The number of buffer areas to be allocated. For a file which is to be processed at the logical level this must be one or two. For a file which is only to be processed at the physical level this should be zero.

L  =  The length in words of each buffer area. If N is one or two, L must be equal to the bucket size of the file. If N is zero L should be zero.

FDA  =  The start address of the File Definition Area.

LABEL  =  The instruction in the user program to which control will be transferred should an exception condition arise. The different types of errors and exceptions are distinguished by codes put in words 10 and 11 of the program.

MODE  =  The mode of opening the file. This will be represented by a number in the range 1 to 9. See Note below for the meaning of these numbers.
PROC = The type of processing to be undertaken. S = Serial processing, Q = Sequential or random. Serial processing implies that tags are not followed to overflow buckets, when processing at the logical level. At the physical level this parameter is ignored since tags are never followed.

O = Overflow buffer indicator. For a file which is being processed at the logical level, and for which standard housekeeping overflow procedures are being used, the housekeeping system must have the use of a buffer area in store to hold overflow buckets. This can be one of the home bucket buffers specified by N, in which case O will be omitted. If an extra buffer is being assigned specifically for overflow an alphabetic O will be written in this field. For all other conditions O should be omitted.

L1 = Overflow indicator. This applies only to files opened in overlay mode (decimal coded modes 1, 2, or 3). If Housekeeping routines for first level overflow only are required, L1 set to 1. If Housekeeping routines for second level overflow are also required, L1 is set to the length of the longest record in the file. If no Housekeeping overflow routines are required, L1 is omitted (see under LABEL1 below).

LABEL1 = The instruction in the user program to which control will be transferred if bucket overflow occurs. This implies that the user has written his own routine for handling bucket overflow. If it is desired to use the housekeeping routine provided, this field should be omitted. If, however, both L1 and LABEL1 are omitted it is assumed that overflow cannot occur and no overflow routines will be used. The parameter applies only to files opened in overlay mode (decimal coded modes 1, 2 or 3).

Notes
1 The mode entered in the operand field of an SDDEF instruction must be a number in the range 1 to 9. The following table shows the opening mode that each number represents:

<table>
<thead>
<tr>
<th>Decimal Coded Mode</th>
<th>Magnetic Tape Mode</th>
<th>Direct Access Device Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#200</td>
<td>#100</td>
</tr>
<tr>
<td>2</td>
<td>#300</td>
<td>#100</td>
</tr>
<tr>
<td>3</td>
<td>#400</td>
<td>#100</td>
</tr>
<tr>
<td>4</td>
<td>#200</td>
<td>#200</td>
</tr>
<tr>
<td>5</td>
<td>#300</td>
<td>#300</td>
</tr>
<tr>
<td>6</td>
<td>#400</td>
<td>#300</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>#400</td>
</tr>
<tr>
<td>8</td>
<td>#600</td>
<td>#600</td>
</tr>
<tr>
<td>9</td>
<td>#100</td>
<td>#200</td>
</tr>
</tbody>
</table>
Plan 3.4

Delete Record

Magnetic Tape and Cassette Tape
This SD macro-instruction refers only to direct access devices, and causes no action on cassette and magnetic tape devices.

Direct Access

Function Deletes an existing record from a file.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDDEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>_operand</td>
<td>Blank.</td>
</tr>
</tbody>
</table>

Notes
1. In the automatic mode the SDDEL instruction deletes from the file the last record read.
2. In the non-automatic mode the SDDEL instruction deletes from the file the record specified.
SDEND

Close File

Magnetic Tape and Cassette Tape

Function Terminates the processing of either input or output files.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDEND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>This may be any one of: REWIND, CLOSE, UNLOAD, or LEAVE (not PLAN 1).</td>
</tr>
</tbody>
</table>

Notes

1. REWIND = tape rewound or cassette aligned, unit retained by program.
2. CLOSE = tape rewound or cassette aligned, unit disengaged but not disconnected. In PLAN 1 the cassette is aligned and the unit disengaged and disconnected; that is, the action is the same as for UNLOAD.
3. UNLOAD = tape rewound or cassette aligned, unit disengaged and disconnected.
4. LEAVE = tape is not rewound or cassette aligned, unit retained by program.

Direct Access

Function Terminates the processing of input, output or overlay files.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDEND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>This may be either CLOSE or RESET</td>
</tr>
</tbody>
</table>

Notes

1. CLOSE = deletion of the file from the program's reservations.
2. RESET = file still available to the program.

2. RESET will be compiled as REWIND in a magnetic or cassette tape mode.
SDEXT

PLAN 3,4

Extend or Contract a File

Magnetic Tape and Cassette Tape
This SD macro-instruction refers only to direct access devices, and causes no action on cassette and magnetic tape devices.

Direct Access

Function
Allocates additional backing storage space to an output file, or reduces its size by an integral number of file areas, during the running of a program.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>B,M</td>
</tr>
</tbody>
</table>

B may be:

1. A decimal integer which is eight or a multiple of eight.
   In this form, B specifies the number of blocks by which the file is to be extended. The number must not be negative.

2. A decimal integer in the range 1 to 7.
   In this form, B refers to an accumulator which contains a signed binary number. If the number is positive it specifies the number of blocks by which the file is to be extended. If the number is negative it specifies the number of blocks by which the file is to be contracted.

M may be:

1. The serial number of the cartridge, unit or magazine on which Executive is to attempt to extend the file. The serial number must be an octal number in the range #1 to #777777.

2. A reference to an accumulator which contains the serial number of the cartridge, unit or magazine on which Executive is to attempt to extend the file. The accumulator must not be X0. The reference is written as a seven digit octal number comprising the accumulator number and six trailing digits. (The value of the trailing digits is immaterial.)

3. Omitted, together with its preceding comma. M should always be omitted if contraction of the file is required. If extension of the file is required and M is omitted, Executive will attempt to extend the file on any cartridges, units or magazines that are on line.

Notes

1. If a file is successfully extended, it will be extended by one file area having a number of blocks equal to or greater than the number requested.

2. If a file is successfully contracted, it will be contracted by an integral number of file areas having in total a number of blocks equal to or less than the number specified. File areas are removed from the end of the file until the removal of a further file area would result in the contraction of the file by a greater number of blocks than that specified.

3. The SDEXT macro-instruction may only be used for files opened in output, end-of-file or scratch modes (decimal coded modes 5, 6, 7 or 8).

4. If M is expressed as zero it will be treated as though M were omitted.
SDFAB

Create a Table of File Area Boundaries (U.D.A.S. Only)  PLAN 3,4

Magnetic Tape and Cassette Tape

This SD macro-instruction refers only to direct access devices, and causes no action on cassette and magnetic tape devices.

Direct Access Function

Creates a table of file area boundaries. It may only be used with devices that conform to the unified direct access standards.

Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>SDFAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File unit number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M), L, S</td>
</tr>
</tbody>
</table>

N may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

N(M) refers to a location which is the first word of an area in which the table of file area boundaries will be created.

L may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2 An absolute expression in the range 0 to 7, e.g.
   (a) A decimal integer in the range 0 to 7.
   (b) An octal integer in the range #0 to #7.
   (c) A previously defined absolute symbol with a value in the range 0 to 7.

3 A literal.

L refers to a location which contains the size of the area whose start address is specified by N(M).

S may be the character S, or may be omitted. If it is present, the user will be informed each time processing is about to occur on a different direct access storage unit. If it is omitted, no indication will be given that processing is about to occur on a different unit.

Notes

1 The table created by SDFAB will be analysed each time a bucket is about to be read. If this bucket resides in the current file area no action is taken. However, if the current read requires another file area to be
accessed all the updated buckets held in the Housekeeping buffers will be written back to the file.

If 'S' is omitted no further action occurs. If 'S' is present and the current read requires another unit to be
accessed, the user will be informed of the impending unit change by exception condition #01 (bits 6 to 23
of Word 11 will be zero with this exception). As usual, to re-enter the routine in which this exception
occurred the user must branch to the address held in word 16 of the File Definition Area currently being
processed.

2 Extensions and contractions should be performed through the use of SDEXT, as in this manner the SDFAB
table will be updated to reflect the new state of the file areas. If a bucket to be used is not defined by any of
the file area descriptions contained in the SDFAB table no action is taken.

3 The length of the area for the creation of the table should be two words per file area plus four words.

4 This macro has no effect when Physical processing is being used.

Examples

SDFAB 2  BU'affa(1), '124', S
SDFAB 12  BU'affa,SIZE
SDFAB 7  BU'affa(3), '104'
SDFES

Skip Forward to End-of-Subfile Sentinel

Cassette Tape and Direct Access

This SD macro-instruction refers only to magnetic tape, and causes no action on cassette tape and direct access devices.

Magnetic Tape

Function  Moves a magnetic tape forwards until the end-of-subfile sentinel of the specified subfile is found, and leaves the tape positioned so that the read head is on the trailer label side of the sentinel.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDFES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
<tr>
<td></td>
<td>N may be:</td>
</tr>
<tr>
<td></td>
<td>1 A relative expression which refers to a lower data location, e.g.</td>
</tr>
<tr>
<td></td>
<td>(a) A symbolic name referring to a lower data location.</td>
</tr>
<tr>
<td></td>
<td>(b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.</td>
</tr>
<tr>
<td></td>
<td>2 An absolute expression in the range 0 to 4095, e.g.</td>
</tr>
<tr>
<td></td>
<td>(a) A decimal integer in the range 0 to 4095.</td>
</tr>
<tr>
<td></td>
<td>(b) An octal integer in the range #0 to #7777.</td>
</tr>
<tr>
<td></td>
<td>(c) A previously defined absolute symbol with a value in the range 0 to 4095.</td>
</tr>
</tbody>
</table>

The operand N(M) refers to a location which is the first word of the 20 word Subfile Definition Area.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Notes

1 The SDFES macro-instruction may be used only for processing composite files.
2 The subfile specified in the Subfile Definition Area may be a simple or a composite subfile and may be at any level.
3 The SDFES macro-instruction is not accepted by the compilers XPLR and XPLS.
SDFSS

PLAN 3, 4

Cassette Tape and Direct Access

This SD macro-instruction refers only to magnetic tape, and causes no action on cassette tape and direct access devices.

Magnetic Tape

Function

Moves a magnetic tape forwards until the specified start-of-subfile sentinel is found, and leaves the tape positioned so that the read head is on the trailer label side of the sentinel.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDFSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) refers to a location which is the first word of the 20 word Subfile Definition Area.

Modification

This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Notes

1. The SDFSS macro-instruction may be used only for processing composite files.

2. The subfile specified in the Subfile Definition Area may be a simple or a composite subfile and may be at any level.

3. The Subfile Definition Area must initially contain words 2 to 6 of the start-of-subfile sentinel. When the pseudo-operation has been completed the Subfile Definition Area contains the 20 words from the qualifier block of the specified start-of-data sentinel.

4. In the case of a multi-reel file, unless this is the first instruction other than SDDEF to address the file, Housekeeping commences search for the specified sentinel from the current position on the tape, and opens continuation reels if necessary. If this is the first instruction other than SDDEF to address the file, Housekeeping may go directly to the required reel of the multi-reel file if the magnetic tape is suitably specified in the File Definition Area.

5. The SDFSS macro-instruction is not accepted by the compilers #XPLR and #XPLS.
Skip Forward to User Sentinel

Cassette Tape and Direct Access

This SD macro-instruction refers only to magnetic tape and causes no action on cassette tape and direct access devices.

Magnetic Tape

Function

Moves a magnetic tape forwards, leaving it positioned with the read head on the trailer label side of the first user sentinel encountered after a specified number of intervening user sentinels has been ignored.

Format

Operation Code

SDFTS

Accumulator

File number. This must be a decimal integer in the range 0 to 15.

Operand

N(M)

N must be a decimal integer in the range 0 to 4095.

The operand N(M) specifies the number of user sentinels to be ignored.

Modification

This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Notes

1. If N(M) is zero or is omitted, the tape is moved forward and is left positioned with the read head on the trailer label side of the first user sentinel encountered.

2. The SDFTS macro-instruction may be used for processing a simple file or for processing a simple subfile within a composite file.
SDIND

PLAN 3,4

Search Index Tables

Magnetic Tape and Cassette Tape
This SD macro-instruction refers only to direct access devices, and causes no action on cassette or magnetic tape.

Direct Access

Function Examines standard index tables to identify the bucket containing the specified record.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDIND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>A1, A2 (both optional), where A1 and A2 are the start addresses of areas in store (in lower or upper data) previously allocated to hold index tables.</td>
</tr>
</tbody>
</table>

Notes

1. The operand field of an SDIND instruction may be blank or may contain any of the following entries.
   - A1, A2
   - A2
   - 0, A2
   - A1
Cassette Tape and Direct Access

This SD macro-instruction refers only to magnetic tape, and causes no action on cassette tape and direct access devices.

Magnetic Tape

Function
Relabels an output magnetic tape.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M),L</td>
</tr>
</tbody>
</table>

N may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

N(M) refers to a location which is the first word of a parameter table.

L may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2 An absolute expression in the range 0 to 7, e.g.
   (a) A decimal integer in the range 0 to 7.
   (b) An octal integer in the range #0 to #7.
   (c) A previously defined absolute symbol with a value in the range 0 to 7.

3 A literal.

L refers to a location which contains the number of entries in the parameter table.

Modification
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the first term of the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the first term of the operand.

Notes

1 The SDLAB macro-instruction may be used for processing simple or composite files.

2 The SDLAB macro-instruction should be the first instruction other than SDDEF to address the file, if relabelling is required; except that if user's buffers are set up for the file, the SDBUF macro-instruction may either precede or follow the SDLAB macro-instruction.
The magnetic tape that is relabelled is the first available tape that has not already been relabelled for which an entry appears in the parameter table. The parameter table contains an entry for each magnetic tape that is to be relabelled. An entry consists of four words as follows:

<table>
<thead>
<tr>
<th>Words 0 to 2</th>
<th>Existing file name of magnetic tape to be relabelled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 3</td>
<td>Bits 0 to 2: Undefined.</td>
</tr>
<tr>
<td></td>
<td>Bits 3 to 11: Existing reel sequence number.</td>
</tr>
<tr>
<td></td>
<td>Bits 12 to 23: Existing file generation number.</td>
</tr>
</tbody>
</table>

The table may be any length and is terminated by a word containing four asterisks.

The entries in the table may be preset, or may be inserted at run time from paper tape or cards by use of the subroutine HPAR in compact mode programs or HPARE in extended data mode programs, as described in the manual 'Magnetic Tape', Chapter 11.

The file name, reel sequence number, file generation number and retention period to be written to the relabelled tape are obtained from the File Definition Area. These parameters may be preset, or may be inserted at run time from paper tape or cards by use of the subroutine HPAR in compact mode programs or HPARE in extended data mode programs, as described in the manual 'Magnetic Tape', Chapter 11.

The decimal coded opening mode specified in the SDDEF macro-instruction should be 6.

The SDLAB macro-instruction is not accepted by the compilers #XPLR and #XPLS.
SDRD

Read a Record

Magnetic Tape, Cassette Tape and Direct Access

Function Transfers one record from a housekeeping buffer area to an area specified by the user.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDRD</th>
</tr>
</thead>
</table>

Accumulator File number. This must be a decimal integer in the range 0 to 15.

Operand N(M)

N may be

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) refers to a location which is the first word of an area in the user's processing area to which the record is to be transferred.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.
PLAN 3.4
(and PLAN 1 for Cassette Tape)

Magnetic Tape, Cassette Tape and Direct Access

Function  Reads part or all of a magnetic or cassette tape block or direct access bucket, from the device into core store.

Format  

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDRDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M), L, C</td>
</tr>
</tbody>
</table>

N may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

N(M) refers to a location which is the first word of a buffer area.

L may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2 An absolute expression in the range 0 to 7, e.g.
   (a) A decimal integer in the range 0 to 7.
   (b) An octal integer in the range #0 to #7.
   (c) A previously defined absolute symbol with a value in the range 0 to 7.

3 A literal.

L refers to a location which contains the number of words to be read. This number must be less than or equal to the number of words in the maximum sized block or bucket.

C may be one of the three characters S, 2 or 3, or may be omitted.

The C field of the operand, if present, modifies action on exception conditions in the manner described in note 1 below.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the first term of the operand. In the extended-data mode, the least significant 22 bits of N + M are taken as the first term of the operand.
Notes

1. In the absence of the C field the program is suspended until the transfer is completed, normal Executive action takes place in the event of a transfer failure, and control passes to the user exception routine if an exception condition occurs. If the C field is present, action depends on which parameter, S, 2 or 3, is used, as follows:

S: The transfer is initiated but the program is not suspended, control passing to the user’s program. The transfer is not checked by Housekeeping; except that if a further SDRDB instruction is given before the transfer is completed, then, whether that instruction has an S parameter present or not, the program is suspended until the former transfer is completed, the transfer is checked by Housekeeping, and, if no exception condition has occurred, the further transfer is then initiated.

2: In cassette tape mode or direct access device mode this parameter has no effect.

In magnetic tape mode this parameter specifies that additive mode #20000 is to be used for the transfer. If a transfer failure is detected control is passed to the user exception routine in the usual way.

3: As for 2, except that in magnetic tape mode additive mode #30000 is used.

If the S parameter is used then, unless a further SDRDB instruction follows before the transfer is completed (see above), an SDSUS instruction should subsequently be given in order to check that the transfer has terminated successfully. The buffer area specified should not be accessed before the transfer has been checked.

2. With the compilers #XPLR and #XPLS the C field may contain S or may be absent, but the parameters 2 or 3 are not accepted.
PLAN 3,4  
(and PLAN 1 for Cassette Tape)

Magnetic Tape, Cassette Tape and Direct Access

Function  Places in a user-specified location the address of the location in the Housekeeping buffer area that holds the first word of the next record to be processed.

Format  

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDRDP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Accumulator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N(M)</td>
<td>N may be:</td>
</tr>
<tr>
<td></td>
<td>1 A relative expression which refers to a lower data location, e.g.</td>
</tr>
<tr>
<td></td>
<td>(a) A symbolic name referring to a lower data location.</td>
</tr>
<tr>
<td></td>
<td>(b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.</td>
</tr>
<tr>
<td></td>
<td>2 An absolute expression in the range 0 to 4095, e.g.</td>
</tr>
<tr>
<td></td>
<td>(a) A decimal integer in the range 0 to 4095.</td>
</tr>
<tr>
<td></td>
<td>(b) An octal integer in the range #0 to #7777.</td>
</tr>
<tr>
<td></td>
<td>(c) A previously defined absolute symbol in the range 0 to 4095.</td>
</tr>
</tbody>
</table>

The operand N(M) refers to a location into which the start address of the next record to be processed is to be placed.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Notes

1 The required address is placed in the least significant 15 bits of the special location; the contents of the most significant 9 bits are indeterminate. In the extended data mode the required address is placed in the least significant 22 bits of the specified location; the contents of the most significant two bits are indeterminate.

2 The SDRDP macro-instruction enables the user to process records without transferring them from the input buffer. In magnetic tape mode or cassette tape mode the length of the record within the buffer area must not be increased during the processing. In direct access device mode the record must not be updated within the buffer area, and the macro-instruction must only be used to process files opened in input mode (decimal coded modes 4 or 9).
Read Reverse One Block

Cassette Tape and Direct Access
This SD macro-instruction refers only to magnetic tape, and causes no action on cassette tape and direct access devices.

Magnetic Tape

Function
Reads in a reverse direction part or all of a block.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDRRB</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Accumulator File number. This must be a decimal integer in the range 0 to 15.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Operand N(M), L</th>
</tr>
</thead>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

N(M) refers to a location which is the first word of a buffer area.

L may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 7, e.g.
   (a) A decimal integer in the range 0 to 7.
   (b) An octal integer in the range #0 to #7.
   (c) A previously defined absolute symbol with a value in the range 0 to 7.

3. A literal

L refers to a location which contains the number of words to be read. This number must be less than or equal to the number of words in the maximum sized block.

Modification This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the first term of the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the first term of the operand.
Notes

1. Words are read into the buffer area starting at the right-hand end.
2. This instruction may only be used when the tape unit addressed has a hardware facility for reading in the reverse direction.
3. The SDRRB macro-instruction must not be used for processing composite files.
SDSUS

Check Previous Transfer to File

Magnetic Tape, Cassette Tape and Direct Access

Function

Checks for the successful completion of otherwise unchecked SDRDB and SDWRB transfers.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDSUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>Blank</td>
</tr>
</tbody>
</table>

Notes

1. The SDSUS macro-instruction is the Housekeeping equivalent of a SUSBY (which cannot be used as an alternative). The macro-instruction is used only after an SDRDB or SDWRB macro-instruction which has an S parameter in the operand. It suspends the program until the transfer initiated by the previous SDRDB or SDWRB macro-instruction addressing the same file has been completed, and checks whether the transfer was successful.
PLANO 3.4
(and PLAN 1 for Cassette Tape)

Magnetic Tape, Cassette Tape and Direct Access

Function Transfers one updated record to a housekeeping buffer area.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDWR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) refers to a location which contains the first word of the record to be transferred.

Notes

1. In magnetic tape or cassette tape modes there is no difference in function between SDWR, SDWRI and SDWRU. Only if a storage device compatible program is to be written need the tape user consider the differences in the direct access mode between these SD macros.

2. In direct access device mode, SDWR may only be used to write a record which has previously been read from a file. To write a new record SDWRI must be used.
Write (all or part of) a Block or Bucket

(Plan 3.4
(and Plan 1 for Cassette Tape)

Magnetic Tape, Cassette Tape and Direct Access

**Function**
Writes part or all of a magnetic or cassette tape block or direct access bucket, from core store to the device.

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDWRB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accumulator</strong></td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td><strong>Operand</strong></td>
<td>N(M), L, C</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   a. A symbolic name referring to a lower data location.
   b. A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   a. A decimal integer in the range 0 to 4095.
   b. An octal integer in the range #0 to #7777.
   c. A previously defined absolute symbol with a value in the range 0 to 4095.

N(M) refers to a location which is the first word of a buffer area.

L may be:

1. A relative expression which refers to a lower data location, e.g.
   a. A symbolic name referring to a lower data location.
   b. A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 7, e.g.
   a. A decimal integer in the range 0 to 7.
   b. An octal integer in the range #0 to #7.
   c. A previously defined absolute symbol with a value in the range 0 to 7.

3. A literal

L refers to a location which contains the number of words to be written. This number must be less than or equal to the number of words in the maximum sized block or bucket.

C may be one of the three characters S, 2 or 3 or may be omitted.

The C field of the operand, if present, modifies action on exception conditions in the manner described in note 1 below.

**Modification**
This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the first term of the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the first term of the operand.

**Notes**

1. In the absence of the C field, the program is suspended until the transfer is completed, normal Executive action takes place in the event of a transfer failure, and control passes to the user exception routine if an exception occurs.
condition occurs. If the C field is present, action depends on which parameter, S, 2 or 3, is used, as follows:

S: The transfer is initiated but the program is not suspended, control passing to the user’s program. The transfer is not checked by Housekeeping; except that if a further SDWRB instruction is given before the transfer is completed, then, whether that instruction has an S parameter present or not, the program is suspended until the former transfer is completed, the transfer is checked by Housekeeping, and, if no exception condition has occurred, the further transfer is then initiated.

2: In cassette tape mode or direct access device mode this parameter has no effect.

In magnetic tape mode this parameter specifies that additive mode #20000 is to be used for the transfer. If a transfer failure is detected control is passed to the user exception routine in the usual way.

3: As for 2, except that in magnetic tape mode additive mode #30000 is used.

If the S parameter is used then, unless a further SDWRB instruction follows before the transfer is completed (see above), an SDSUS instruction should subsequently be given in order to check that the transfer is terminated successfully. The buffer area specified should not be accessed before the transfer has been checked.

2 With the compilers #XPLR and #XPLS the C field may contain S or may be absent, but the parameters 2 or 3 are not accepted.
SDWRI

Insert New Record

(Plan 3, 4
(and Plan 1 for Cassette Tape)

Magnetic Tape, Cassette Tape and Direct Access

Function Transfers one new record to a housekeeping buffer area.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDWRI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1 A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2 An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) refers to a location which contains the first word of the record to be transferred.

Notes

1 In magnetic tape or cassette tape modes there is no difference in function between SDWR, SDWRI and SDWRU. Only if a storage device compatible program is to be written need the tape user consider the differences in the direct access mode between these SD macros.

2 In the direct access mode used on an overlay file, the SDWRI instruction inserts a record either before the last one read (automatic processing) or in a user specified bucket (non-automatic processing).
Cassette Tape and Direct Access
This SD macro-instruction refers only to magnetic tape and causes no action on cassette tape or direct access devices.

Magnetic Tape

Function  Writes a user sentinel to magnetic tape.

Format  

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDWRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol in the range 0 to 4095.

The operand N(M) refers to a location which is the first word of a 16 word area containing user's information to be included in the qualifier block.

Modification  This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.
SDWRU

Write Unchanged Record

Direct Access
This SD macro-instruction refers only to magnetic and cassette tape and causes no action on direct access devices.

Magnetic Tape and Cassette Tape

Function Transfers one unchanged record to a housekeeping buffer area.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDWRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) refers to a location which contains the first word of the record to be transferred.

Notes

1. In magnetic tape or cassette tape modes there is no difference in function between SDWR, SDWRI and SDWRU. Only if a storage device compatible program is to be written need the tape user consider the differences in the direct access mode between these SD macros.
SDWSS

PLAN 3.4

Cassette Tape and Direct Access

This SD macro-instruction refers only to magnetic tape, and causes no action on cassette tape and direct access devices.

Magnetic Tape

Function

Writes a start-of-subfile sentinel on magnetic tape. If the subfile is a simple subfile, optionally also writes a start-of-data sentinel.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDWSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>File number. This must be a decimal integer in the range 0 to 15.</td>
</tr>
<tr>
<td>Operand</td>
<td>N(M)</td>
</tr>
</tbody>
</table>

N may be:

1. A relative expression which refers to a lower data location, e.g.
   (a) A symbolic name referring to a lower data location.
   (b) A symbolic name as in (a) adjusted by following it with a signed decimal or octal integer.

2. An absolute expression in the range 0 to 4095, e.g.
   (a) A decimal integer in the range 0 to 4095.
   (b) An octal integer in the range #0 to #7777.
   (c) A previously defined absolute symbol with a value in the range 0 to 4095.

The operand N(M) refers to a location which is the first word of the 20 word Subfile Definition Area.

Modification

This statement has an M-field. When modified, the least significant 15 bits of N + M are taken as the operand. In the extended data mode, the least significant 22 bits of N + M are taken as the operand.

Notes

1. The SDWSS macro-instruction may be used only for processing composite files.
2. The subfile specified in the Subfile Definition Area may be a simple or a composite subfile, and may be at any level.
3. If the specified subfile is a simple subfile, then after writing the start-of-subfile sentinel the File Definition Area of the composite file is examined. If bit 0 of word 9 is zero, a start-of-data sentinel is written.
4. The SDWSS macro-instruction is not accepted by the compilers #XPLR and #XPLS.
NOTE ON THE MAGNETIC TAPE HOUSEKEEPING SYSTEM

Prior to the introduction of storage device housekeeping the only housekeeping system available was for use with magnetic tape. The user called relevant routines by giving macros that commenced with the letters MT (e.g. MTDEF, MTRD). ICL compilers still accept MT macros, and the SD macros to which these correspond are shown in the Table below. If MT instructions are used there is no need to give a #CMODE directive in a program, but it will not be possible to write storage device compatible programs.

<table>
<thead>
<tr>
<th>SD macros that cause action in magnetic tape mode</th>
<th>Equivalent MT macros</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDBSS</td>
<td>MTBSS</td>
</tr>
<tr>
<td>SDBTS</td>
<td>MTBTS</td>
</tr>
<tr>
<td>SDBUF</td>
<td>MTBUF</td>
</tr>
<tr>
<td>SDCLB</td>
<td>MTCLB</td>
</tr>
<tr>
<td>SDCLS</td>
<td>MTCLS</td>
</tr>
<tr>
<td>SDCRE</td>
<td>MTCRE</td>
</tr>
<tr>
<td>SDDEF</td>
<td>MTDEF</td>
</tr>
<tr>
<td>SDEND</td>
<td>MTEND</td>
</tr>
<tr>
<td>SDFES</td>
<td>MTFES</td>
</tr>
<tr>
<td>SDFSS</td>
<td>MTFSS</td>
</tr>
<tr>
<td>SDFTS</td>
<td>MTFTS</td>
</tr>
<tr>
<td>SDLAB</td>
<td>MLAB</td>
</tr>
<tr>
<td>SDRD</td>
<td>MTRD</td>
</tr>
<tr>
<td>SDRDB</td>
<td>MTRDB</td>
</tr>
<tr>
<td>SDRDP</td>
<td>MTRDP</td>
</tr>
<tr>
<td>SDRRB</td>
<td>MTRRB</td>
</tr>
<tr>
<td>SDSUS</td>
<td>MTSUS</td>
</tr>
<tr>
<td>SDWR</td>
<td>MTWR</td>
</tr>
<tr>
<td>SDWRB</td>
<td>MTWR</td>
</tr>
<tr>
<td>SDWR1</td>
<td>MTWR</td>
</tr>
<tr>
<td>SDWRS</td>
<td>MTWRS</td>
</tr>
<tr>
<td>SDWRU</td>
<td>MTWR</td>
</tr>
<tr>
<td>SDWSS</td>
<td>MTWSS</td>
</tr>
</tbody>
</table>

MTBSS, MTBUF, MTCLS, MTFES, MTFSS, MLAB and MTWSS are not accepted by the cassette tape compilers #XPLR and #XPLS.

PLAN 4 compilers provide the compact mode or the mode-compatible versions of the housekeeping routines in response to MT macros in the same way as they do in response to SD macros.
OVERLAY MACRO-INSTRUCTIONS

Three overlay macro-instructions are available to the programmer for the purposes of communication between units of an overlay program. They are:

ENTER
RECAL
BRING

When the PLAN3 or PLAN4 compiler encounters any of these pseudo-operations, it generates the appropriate CALL to a cue within the run-time overlay package, together with the appropriate parameters. All the run-time overlay packages contain the same cuename; the operand of the generated CALL is this cuename and not the subroutine segment name: this makes it unnecessary for the programmer to decide at the time when he is writing the overlay macro-instructions, which overlay medium is to be used. Consolidation of the run-time package (from the library subroutine tape) for the desired overlay medium is achieved by specifying the name of the package under the #PERMANENT directive in the steering segment (see Chapter 6, page 31).

In the case of ENTER and RECAL, when the overlay has been read into core (if required), a branch is generated to a cue within the overlay unit.

The required arrangement of segments into overlay units is also defined in the steering segment, by means of the #OVERLAY directive (see Chapter 6, page 29).

The three overlay macro-instructions are described in the following pages in accordance with the format used in Chapter 4.
ENTER

**Function**
Ascertains which overlay unit contains the specified cue; if this overlay unit is already in core store, branches to a specified location in it; otherwise, brings it into store before branching.

**Format**
- Operation Code: ENTER
- Accumulator: Blank
- Operand: CUENAME + N, where CUENAME is the name of the cue in the required overlay unit, and CUENAME + N is the location at which it is to be entered.

**Execution**
The ENTER instruction causes the program at run-time to find the overlay unit containing the specified cue and, if the overlay unit is not already in core store, to bring it in before entering it at the specified location. If the overlay is already in store, there is an immediate branch to the specified point.

The link accumulator used when the run-time overlay package is called is X1.

When the user’s overlaid program is entered the contents of X1 are such that if the instruction

```
EXIT 1 0
```

is obeyed, control will return to the instruction following whichever ENTER instruction caused the link to be set.

The preservation of this link address while the overlaid program is being obeyed is the responsibility of the programmer. One overlay unit may ENTER (or RECAL or BRING) another, but care must be taken over the preservation of links, particularly if one overlay unit overlays itself.

**Notes**

1. When an ENTER instruction is used to bring an overlay unit into core store from magnetic tape or cassette tape, Executive initiates the transfer without checking the state of any peripheral devices, other than the one from which the overlay unit is being read. If, in these circumstances, any other peripheral units are assigned to the program and are active at the time the instruction is given, the result is undefined. This restriction does not apply to magnetic tape rewinds or cassette tape alignments, provided that the reply words for these are not within the area being overlaid.

2. If any flag setting peripheral is assigned to the program, the result of using an ENTER instruction to bring an overlay unit into core store from magnetic tape or cassette tape is undefined.
**Function**

Ascertains which overlay unit contains the specified cue, brings it into core store and branches to a specified location in it.

**Format**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>RECAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>CUENAME + N, where CUENAME is the name of the cue in the required overlay unit, and CUENAME + N is the location at which it is to be entered.</td>
</tr>
</tbody>
</table>

**Execution**

The RECAL instruction causes the program at run-time to find the overlay unit containing the specified cue, to bring it into core store and then to enter it at the specified location. This happens whether or not the overlay unit is already in core store, enabling the programmer to reset a unit to its initial conditions after it has been altered in some way.

The link accumulator used when the run-time overlay package is called is X1.

When the user's overlaid program is entered the contents of X1 are such that if the instruction

```
EXIT 1 0
```

is obeyed, control will return to the instruction following whichever RECAL instruction caused the link to be set.

The preservation of this link address while the overlaid program is being obeyed is the responsibility of the programmer. One overlay unit may RECAL (or ENTER or BRING) another, but care must be taken over the preservation of links, particularly if one overlay unit overlays itself.

**Notes**

1. When a RECAL instruction is used to bring an overlay unit into core store from magnetic tape or cassette tape, Executive initiates the transfer without checking the state of any peripheral devices, other than the one from which the overlay unit is being read. If, in these circumstances, any other peripheral units are assigned to the program and are active at the time the instruction is given, the result is undefined. This restriction does not apply to magnetic tape rewinds or cassette tape alignments, provided that the reply words for these are not within the area being overlaid.

2. If any flag setting peripheral is assigned to the program, the result of using a RECAL instruction to bring an overlay unit into core store from magnetic tape or cassette tape is undefined.
BRING

Function
Brings an overlay unit into core store (unless it is already there), but does not enter it.

Format
Operation Code  | BRING
Accumulator     | Blank
Operand         | CUENAME, where CUENAME is the name of a cue in the required overlay unit.

Execution
The BRING instruction causes the overlay unit containing the specified cue to be brought into core store if it is not already there, but the program does not enter the unit. If the overlay unit is already in core store, the program continues immediately with the instruction following the BRING instruction; if the overlay unit is not already in core store, the program continues with the instruction following the BRING instruction after the reading in of the overlay unit has been completed.

The link accumulator used is X1, whose contents are destroyed by the BRING instruction. The state of V and the mode register is preserved.

Notes
1. When a BRING instruction is used to bring an overlay unit into core store from magnetic tape or cassette tape, Executive initiates the transfer without checking the state of any peripheral devices, other than the one from which the overlay unit is being read. If, in these circumstances, any other peripheral units are assigned to the program and are active at the time the instruction is given, the result is undefined. This restriction does not apply to magnetic tape rewinds or cassette tape alignments, provided that the reply words for these are not within the area being overlaid.

2. If any flag setting peripheral is assigned to the program, the result of using a BRING instruction to bring an overlay unit into core store from magnetic tape or cassette tape is undefined.
THE DUMP AND RESTART PACKAGE

The pseudo-operation associated with the Dump and Restart package enables the user to incorporate routines providing automatic dumps at any points in his program.

The macro-instruction statement, SDUMP, is described on the next page in accordance with the format used in Chapter 4.
SDUMP

Function

Dumps the contents of the program area, together with information about all basic peripherals and storage device files currently allocated to the program, onto a specified storage device file.

Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>SDUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>Blank</td>
</tr>
<tr>
<td>Operand</td>
<td>A, B</td>
</tr>
</tbody>
</table>

A must be a symbolic name referring to a location in upper or lower data which contains the first word of the dump parameter table.

B is a list of the file numbers of all files onto which the dump may be required, and/or a list of the names of subroutines controlling dumping onto each type of device on which a dump may be required; the items of the list being separated by commas.

Notes

1. If the SDUMP macro-instruction is used in a program, all files processed by the program must be processed by the Storage Device Housekeeping system.

2. The file onto which the dump takes place is determined by the first word of the dump parameter table addressed by A.

3. If subroutine names are included in the list B, only names permitted by the specification of the Dump and Restart package may be included.

4. A call is generated for each item in the list B, so the items in the list should be kept to a minimum compatible with the degree of flexibility required.

5. The format of the dump parameter table and a specification of the Dump and Restart package may be found in the manuals 'Magnetic Tape', 'Cassette Tape' and 'Direct Access'.
Chapter 6  Directives

INTRODUCTION
This chapter contains a description of each of the PLAN directive statements. They are dealt with in the body of the chapter in alphabetical order for purposes of easy reference.

General Description of Directives
A directive gives a compiler information about the organization of a program. For example the #END directive indicates that an entire program segment has been read and causes the compiler to take the appropriate action.

A directive may be either 'major' or 'minor' (a minor directive is also sometimes termed an 'interrupt' directive).

A major directive causes the compiler to treat succeeding statements in a certain way and to continue to do so until another major directive is encountered. Thus, each major directive cancels the effect of the previous major directive.

The major directives are:
  #CMODE
  #ELASTIC
  #END
  #FINISH
  #LOWER
  #MACRO
  #OMIT
  #OVERLAY
  #PERIPHERAL
  #PERMANENT
  #LOWER
  #PMODE
  #PROGRAM
  #UPPER
  #STOP

A minor or interrupt directive relates to a small, prescribed number of succeeding statements. After these few statements have been compiled, the last major directive encountered takes over again.

Minor directives are divided into two categories, program area directives and general purpose directives. The former may appear only in those parts of a program written under the #PROGRAM major directive, while the latter may appear anywhere in a program.

The program area directives are:
  #CUE
  #ENTRY
  #MONITOR
The general purpose directives are:

#COMPLETE
#
#DEFINE
#ERRORSEG
#HMODE
#LIBRARY
#ORDER
#OUST
#PAGE
#SET
#SWITCH

The name of a directive statement is written in the label field of the PLAN coding sheet, and is always preceded by the symbol #. Thus, if the name contains more than five characters it will continue into the operation field, which is otherwise left blank. The accumulator field is always left blank.

For some directives, parameters are written in the operand field of the coding sheet. For other directives, this field is left blank.

Full details for writing each directive statement are given in the individual descriptions which follow.

STEERING SEGMENTS

The descriptions of some of the PLAN directive statements specify that the directive concerned must be in the steering segment. A steering segment, where one is required, is a segment which must be the first segment of a program and which must contain only permitted PLAN directives, i.e. directives which, in their individual descriptions, are stated as being appropriate to a steering segment.

Steering Segments for Overlays

The steering segment must contain as a minimum a #OVERLAY directive which specifies the overlay areas and units to which the segments belong, and a #PERMANENT directive which specifies the name of the run time package required. Run time packages are unique to specific storage media and determine the medium on which the whole program is to be held. The #PERMANENT directive may also be used to specify segments to be held in permanent storage. If data is required by more than one segment of an overlay then the steering segment will contain #LOWER or #UPPER directives which specify OVERCOMMON storage. Another directive which may appear in the steering segment is #OMIT, which enables overlay programs to be compiled with segments missing.

DESCRIPTIONS OF PLAN DIRECTIVES

Each PLAN directive statement is dealt with in its own subsection in the following pages. The directives are in alphabetical order and the description of each starts on a fresh page. The versions of PLAN with which a directive is available and whether it is a major or minor directive are indicated at the head of the first page of each description.
Function

The `#CMODE` directive (Compatible Mode Steering Directive) defines to the compiler which files are to be found on which types of storage devices. The information is required by a PLAN 3 or PLAN 4 compiler when it encounters SD macro-instructions.

Format

`#CMODE` is written in the label field. The first and following lines of the operand field are used to list storage device codes with the appropriate file numbers in parentheses after each one. The entry takes the form:

```
DEVICE CODE (FILE NOS.) DEVICE CODE (FILE NOS.) etc.
```

The device codes which may currently be used are:

- **EDS** Exchangeable Disc Store
- **FDS** Fixed Disc Store
- **MCF** Magnetic Card File
- **MT** Magnetic Tape
- **CTH** Cassette Tape (See Notes 3 and 4).
- **DA** UDAS Direct Access Store

All the files opened in a program must be specified by a file number (in the range 0 to 15) either as being on a particular device or as NULL. The file numbers must be defined either as an inclusive range (e.g. 3 - 7) or as single integers separated by commas (e.g. 2, 8, 12).

Example

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| #CMODE | EDS(0,1) | MT(2-4,5) | MCF(6,7) | NULL | (8-10) |     |     |     |     |     |     |     |     |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

Notes

1. If NULL is written followed by file numbers, the section of the program referring to those files will not be compiled.
2. Only one `#CMODE` directive can be accepted during the compilation of any program. In the case of a storage device compatible program this directive should be inserted in a steering segment, so that only this segment has to be altered if the storage device configuration is changed. In the case of an overlay program the `#CMODE` directive could be given in the overlay steering segment.
3. There are two versions of the Cassette Tape Housekeeping macros which can be called by PLAN 3 and PLAN 4 compilers; one version suitable for use on machines with not less than 8K core store and having a console typewriter, the other version suitable for use on machines with not less than 8K core store without a console typewriter. The compiler will always set itself initially to output the version applicable to the environment in which it is working, but the option exists of specifying the alternative version, by use of the appropriate device code. The relevant device codes are interpreted as follows:

- **CT** The version of Cassette Tape Housekeeping macros suitable for use in the environment in which the compiler is working is output. This device code may safely be used where programs are compiled on the machines on which they are to be run; but care should be exercised if they are compiled elsewhere.
- **CTH** The version suitable for use on machines without a console typewriter ('handswitch' machines) will be output.
- **CTT** The version suitable for use on machines with a console typewriter will be output.
It is possible to use more than one of these three device codes in the same #CMODE directive, with different associated file numbers; but this would not normally be done.

There is a third version of Cassette Tape Housekeeping macros which is output by the PLAN 1 compiler #XPLQ. This compiler (which will list but will not otherwise action a #CMODE directive) outputs a version suitable for use on 4K machines without a console typewriter. If advantage is being taken of remote testing facilities to compile cassette tape programs which are to be run eventually in a 4K machine, the program should be written in PLAN 1 and compilation by #XPLQ should be specified.
(Minor) PLAN 1,2,3,4

Function

The \# directive (the 'Comment' directive) instructs the compiler that the symbols appearing in the operand field are comments only. These comments will be listed in the compiler printout, but will have no effect on the object program.

Format

The directive symbol \# is placed in the first column of the label field, the remainder of which is left blank, as are the whole of both the operation and accumulator fields. The comments are written in the operand field. If the comments extend over more than one line, then each of those lines must be in the format here defined.

Notes

1 Comments may also be written alongside other statements. The comments are written in upper case letters, and each must be preceded by a left-hand square bracket, i.e. [, which signals to the compiler that a comment is intended.

In PLAN 1 only, there must be a blank space between the statement and the comment, though this is desirable in PLAN 2, 3 and 4 also, so as to improve legibility. When possible, the square bracket is conventionally written in column 36 of the coding sheet, with the comment immediately following. This column is marked by the symbol† printed at the top of the coding sheet.

Comments written as here defined will have no effect on the object program; but if comment preceded by a left-hand square bracket is written on a line by itself, that is, not alongside a PLAN statement, then it will cause a zero word to be generated in the object program.
# COMPLETE

Function
The #COMPLETE directive instructs the compiler that it is dealing with a program that does not require the inclusion of standard library subroutines. The library will not, therefore, be searched even if there are blank cues in the program.

Format
#COMPLETE is written in the label field, extending into the operation field. The rest of the line is left blank.

Notes
1. 'Standard library subroutines' includes any subroutines read from the library tape including those required for I/O Generator, M.T.H., overlay packages, Monitor, etc.

2. The #COMPLETE directive is effective only when the program is compiled in any of the following circumstances:
   (a) by a PLAN 4 compiler.
   (b) by a PLAN 3 compiler which has a built-in consolidation facility.
   (c) by a PLAN 3 compiler with output on disc, provided that the automatic consolidation facility is used.

   If the directive is encountered in a compilation by #XPLE, by #XPLR, or by a PLAN 3 compiler with output on disc when the automatic consolidation facility is not in use, then it is listed (provided that LIST or SHORTLIST is present in the steering information) but no other action is taken.

3. If a #COMPLETE directive is present in the program and a #LIBRARY directive (for compilers with output on magnetic tape) or a LIBE or LIBF parameter (for compilers with output on disc) is also present, then the #COMPLETE directive overrides the latter directive or parameter, and no libraries are searched.

4. A #COMPLETE directive may, but need not necessarily, be used in a steering segment.
Function

The #CUE directive instructs the compiler that the symbols which appear in the operand field represent the name by which the next location is to be referenced from other segments of the program.

In PLAN 4, the #CUE directive may additionally be used to generate extra replacers. (Replacers are described in Chapter 1 page 8, and the generation of them is discussed in Chapter 8, page 32.)

Format

#CUE is written in the label field. The operation and accumulator fields are left blank. The cue name is written in the operand field, and may consist of up to eleven alphanumeric characters, of which the first must be alphabetic.

In PLAN 4, the cue name may be followed by parentheses enclosing an absolute expression (for example, a decimal integer, an octal integer or a previously defined symbol) with a value in the range 0 to 510. See under #DEFINE for the rules for writing absolute expressions.

The absolute expression, if present, specifies the number of consecutive locations, commencing with the one immediately following the cued location, for which replacers are to be generated. Such replacers are stored consecutively with the one generated for the cued location, which is automatically provided when the segment is compiled to operate in extended branch mode.

If the segment is consolidated into a direct branch mode program, the specification contained in any absolute expression following a cue name in a #CUE directive is ignored.

Notes

1. An instruction under the #CUE directive may be given a separate label for use within that segment. This label may be, but is not necessarily, the same as the cue name. It is not recommended that an instruction be referenced within the same segment by a cue name unless this name also appears in the label field of the instruction.

2. Any location in the program operation store may be assigned a name by a #CUE directive.

3. In PLAN 4, if the cue name is followed by an absolute expression enclosed in parentheses, and the absolute expression has a negative value or a value greater than 510, then the statement is flagged as a J class error and 510 replacers are generated.
Function

The #DEFINE directive instructs the compiler that the group of characters appearing before the equals sign in the operand field is to be assigned a value equal to that of the group of characters appearing after the equals sign.

Format

#DEFINE is written in the label field, extending into the operation field, and the parameters of the expression are written in the operand field, as follows:

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>48</th>
<th>56</th>
<th>64</th>
<th>72</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>#DEFINE</td>
<td>NAME=EXPRESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The Name may consist of up to eleven alphanumeric characters (or up to five in PLAN 1), of which the first must be alphabetic. The Expression may be either absolute or relative. The rules for both types of expressions are given below.

Rules for Writing Absolute Expressions

An absolute expression may consist of any number of decimal integers, octal integers and symbols.

Each integer or symbol must be preceded by a plus or minus sign or * or /, except in the case of the first term of an expression for which (in the absence of a sign) a plus sign is assumed. The expression is evaluated serially, from left to right.

If a symbol is used in the expression, it must either

1. be associated somewhere in the expression with a symbol of the same group and be preceded by the opposite sign to this accompanying symbol (i.e. the form must be A - B or -A + B, where A and B represent two symbolic identifiers of the same group), or

2. have been set equal to an absolute expression in some previous directive.

In the case of 1 above, symbols are considered to be in the same group if they have been defined under the same #LOWER or #UPPER directive, or if they both appear in the label field of the program area in the same segment.

Rule 1 is due to the fact that the programmer does not directly control the location addresses of words of variable or preset or program areas, but he can regard the difference between the addresses of two symbols of the same group as a fixed (absolute) quantity.

Consider this example:

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>48</th>
<th>56</th>
<th>64</th>
<th>72</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>#DEFINE</td>
<td>NET=GROSSTAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

In this example, NET represents the distance between the two storage locations defined or labelled as GROSS and TAX.

In the case of 2 above, consider these examples:

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>48</th>
<th>56</th>
<th>64</th>
<th>72</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>#DEFINE</td>
<td>NORAH=9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#DEFINE</td>
<td>PATOD=7+NORAH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#DEFINE</td>
<td>AGFIE=8-PATOD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#DEFINE</td>
<td>BERNIE=7.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

In the first example above, the symbol NORAH is given a value of 9. NORAH can then be written in the operand field of any instruction that would normally employ a small integer in that field. It could
be used, for example, in a shift instruction, a load count instruction, or in any instructions that perform arithmetic using small numbers (e.g. LDN, ADN, ORN). The symbol may not be used as a program label, nor may it be used in the accumulator field.

In the second example above NORAH is used as part of the expression defining the symbol PATOD. The expression is valid because NORAH has been previously defined equal to the absolute expression 9. so PATOD has a value of +2.

Similarly the symbol AOIFE is defined in terms of PATOD and has a value of 6.

An expression as defined in 1 or 2 on page 8 may be followed by \( \times \) or \( \div \), where \( I \) represents a decimal or octal integer, \( \times \) denotes 'multiplied by', and \( \div \) denotes 'divided by' (giving an unrounded integral value; the compiler uses the DVS instruction, so the quotient given will be the algebraically next lower integer to the true quotient if the latter is not an integer).

For example:

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td>#DEFINE</td>
<td>WEIGHT1=</td>
<td>SURPLUS-</td>
<td>STANDARD*2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#DEFINE</td>
<td>WEIGHT2=</td>
<td>WEIGHT1*2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#DEFINE</td>
<td>DISTANCE=</td>
<td>39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#DEFINE</td>
<td>VELOCITY=</td>
<td>DISTANCE/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The expressions are evaluated serially from left to right, therefore

\[
\text{WEIGHT1} = \text{SURPLUS-\text{STANDARD}^2}
\]

has the effect \( \text{WEIGHT1} = (\text{SURPLUS-\text{STANDARD}})^2 \)

Rules for Writing Relative Expressions

1. An expression is relative if it contains one positive, non-absolute symbol (i.e. a symbol referring to a location) that does not conform with rules 1 or 2 on page 8, and the expression is otherwise absolute.

For example:

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td>#DEFINE</td>
<td>BLOCKA=</td>
<td>BLOCKB+1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#DEFINE</td>
<td>SIZE1=</td>
<td>TABLE-A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The symbol appearing to the left of the equals sign will subsequently be considered as of the same group as the non-absolute symbol.

2. The expression to the right of the equals sign may also take the following forms:

\[
A * I + R \\
-A * I + R \\
A / I + R \\
-A / I + R,
\]

where \( R \) is any relative expression of the type described in 1 above, and where the left-most part of the formula is evaluated as described in the rules for absolute expressions.
For example:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>#DEFINE</td>
<td>DAVE=REAL+6A+KAY+6k+1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#DEFINE</td>
<td>YEN=ADDLE/3+NAME-476</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The symbol appearing to the left of the equals sign will subsequently be considered to be of the same group as R.

Notes

1. All symbols appearing in expressions must have been previously defined, explicitly or, where permissible, implicitly (see Chapter 2, page 16).

2. The final value of any expression must be in the range 0 to +32,767 (during evaluation, a full 24-bit word is used).

3. In PLAN 2, 3 and 4 the name appearing to the left of the equals sign must not have appeared previously in the segment and cannot be subsequently redefined. In PLAN 1 the name may have been used previously in the segment provided that it has since been deleted from the symbol table by a #OUST directive.

4. Definitions under a #DEFINE directive are applicable only within the segment in which the directive appears.

5. The symbol may not be used in the accumulator field, nor to specify the modifier.

6. A name defined under a #DEFINE directive cannot be used as a parameter for a pseudo-operation.
Function

The #ELASTIC directive designates an upper common variable data block which will be placed at the top of the program's storage area on consolidation, and signifies to the compiler that the succeeding statements are variable data statements requiring storage space in that block.

Format

#ELASTIC is written in the label field, extending into the operation field. The accumulator field is left blank. The entry in the operand field is of the form COMMON/BLOCKNAME/ where BLOCKNAME is a name containing up to eleven alphanumeric characters of which the first must be alphabetic. The following lines of the operand field may contain variable data statements specifying the areas to be allocated storage within the block.

Example

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC.</th>
<th>COMMON</th>
<th>BLOCKNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>#ELASTIC</td>
<td>COMMON</td>
<td>TABLOCK</td>
<td>CONTROL</td>
<td>TABTEST</td>
</tr>
</tbody>
</table>

Notes

1. The rules for writing variable data statements are given in Chapter 2, page 8, under the heading 'Formats of Data Statements'.
2. The block name must be unique within the program, and must not be the same as any segment name or cue name within the program. The area names defined in the variable data statements must be unique within the segment.
3. If the program's core storage area is extended, it is the upper common variable block designated by the #ELASTIC directive which is extended.
4. Only one block may be so designated in a program. If more than one block is defined under #ELASTIC directives, the first one encountered on consolidation is accepted as the designated block, and the others are treated as though they were upper common variable blocks defined under #UPPER directives.
5. The designated block must be defined in each segment which requires to access it. Provided that it is defined under a #ELASTIC directive in one segment, it may be defined in the other segments as a common area under #ELASTIC directives or #UPPER directives.
6. The #ELASTIC directive may, but need not necessarily, appear in a steering segment.
7. If there is any error in the operand field of a #ELASTIC directive then it is treated as a #UPPER directive, and is flagged appropriately in compiler listings (see Chapter 9).
8. Any constant data statements appearing under #ELASTIC directives are flagged as an X class error in compiler listings, but are otherwise ignored.
9. If a storage location is defined under a #ELASTIC directive, then the definition must precede the use of the location's name in the program.
10. Locations defined under a #ELASTIC directive are unlikely to lie below location 4096, and should therefore be addressed only by modified instructions.
11. The block name must not be referenced in the operand field of a non-branch instruction.
12. If a #ELASTIC directive is written thus
the effect is to designate an unnamed area. This area is identified in compiler listings and store analyses by the symbol %.

The facility is intended for use with mixed language programs, the designated area being that which in a FORTRAN segment is the Blank Common Area. The directive would be followed by variable data statements in the usual way.

13 In a paged environment, the #ELASTIC area is considered to be impure.
Function
The #END directive instructs the compiler that this is the final statement of the program segment.

Format
#END is written in the label field and the remainder of the line is left blank.

Notes
1 In the last segment of any program, the directive #END is followed immediately by the directive #FINISH (except in PLAN 1, which caters only for single-segment programs and where #FINISH is not required, and if written is ignored).
#ENTRY

Function

The #ENTRY directive instructs the compiler that the next instruction is the point at which Executive enters the object program for the purpose of starting or restarting the program.

Format

#ENTRY is written in the label field. The appropriate entry point is indicated on the same line by a single digit at the beginning of the operand field.

Notes

1  A maximum of ten #ENTRY directives is permitted in a program. Each entry point is distinguished by writing a unique number (in the range 0 to 9) in the operand field.
2  The directive may appear only in a series of statements under the major directive #PROGRAM.
3  Words 20 to 29, inclusive, of the program are used for the entry points.
4  When the object program has been loaded into the computer, the operator can cause the program to be started at any entry point.
5  At least one entry point should be specified in a program.
Function

The #ERRORSEG directive generates a blank cue, so that the segment specified by it will be called from the program library tape on consolidation. The segment specified will be treated by the consolidator as satisfying all otherwise unsatisfied cues.

Format

#ERRORSEG is written in the label field, extending into the operation field. The accumulator field is left blank. The name of the segment to be called is written in the operand field.

Notes

1. The segment specified by the #ERRORSEG directive must, when in source form, have been headed by the expanded form of the #PROGRAM directive which signifies to the consolidator that the segment is an "error segment", as described on page 39. If not so written, then although called by the consolidator, it will not be treated as satisfying all otherwise unsatisfied cues.

2. If the segment intended as the "error segment" is in the source stream, and hence is not to be called from a program library tape, then it is not necessary to specify it in a #ERRORSEG directive.

3. The #ERRORSEG directive may, but need not necessarily, be used in a steering segment.
Function
The #FINISH directive instructs the compiler that this is the last segment of the program. (For PLAN 1, see Note 2 below.)

Format
#FINISH is written in the label field and extends into the operation field; the remainder of the line is left blank.

Notes
1. In the last program segment, the directive #FINISH follows immediately after the directive #END.
2. With PLAN 1 compilers the #FINISH directive has no effect.
Function

The #HMODE directive is used to specify which version (the compact mode version or the mode-compatible version) of the Storage Device Housekeeping systems and/or the Dump and Restart package is required by the program, where the mode of the required version differs from that specified in the #PMODE directive.

Format

#HMODE is written in the label field. The operation and accumulator fields are left blank. The entry in the operand field is 15AM and DBM if the compact mode version of the related software is required, or 22AM and EBM if the mode-compatible version of the related software is required.

Notes

1 If a #HMODE directive is used, it must be in the steering segment. Only one #HMODE directive is accepted in a program.

2 If there is no #HMODE directive in the steering segment and the program includes calls to the Storage Device Housekeeping systems and/or to the Dump and Restart package, then the address mode specification in the #PMODE directive is taken as specifying also the required version of these software items. Thus, in these circumstances, the compact mode version is supplied unless the #PMODE directive specifies 22AM, in which case the mode-compatible version is supplied.

3 The compact mode version of these routines is designed to operate in 15-bit address mode and direct branch mode. The mode-compatible version is designed to work in either 15-bit address mode or 22-bit address mode, and direct branch mode or extended branch mode.

4 There is no mode-compatible version of the Cassette Tape Housekeeping system, so as regards calls to this system the #HMODE directive has no effect; the 15AM version is supplied, if called, whatever the specification in the #HMODE (or #PMODE) directive may be.
### #LIBRARY

**Function**

The #LIBRARY directive instructs the compiler that it is to search subroutine groups other than or additional to the S-RS group. It is available only with compilers with magnetic tape output.

**Format**

#LIBRARY is written in the label field, extending into the operation field. The accumulator field is left blank. The names of the subroutine groups to be searched are written in the operand field, in the order in which they are to be searched, separated by commas.

**Notes**

1. If no #LIBRARY directive is present in a program, the S-RS subroutine group is searched.
2. If a #LIBRARY directive is present in a program, only the subroutine groups specified by it are searched; so if the S-RS group is among those to be searched, then it must be among those specified.
3. A maximum of six subroutine groups may be specified by the #LIBRARY directive.
4. Each subroutine group name must consist of four characters; if punched with fewer than four characters the compiler will supply spaces at the right-hand end to complete the four characters. Each subroutine group name must commence with a letter; if any subroutine group name specified commences with a non-alphabetic character it is ignored and the line is flagged with a 'D' error flag on the compiler listing, but any other correctly specified subroutine group names in the operand field are accepted.
5. The subroutine groups specified by the #LIBRARY directive must be on the same program library tape as the compiler.
6. The subroutine groups are searched in the order in which they are specified by the #LIBRARY directive, the program library tape being rewound as necessary.
7. Only one #LIBRARY directive may be used in a program. If it appears more than once in a program the second and subsequent appearances are flagged with an 'O' error flag in the compiler listing, and are otherwise ignored.
8. A #LIBRARY directive may, but need not necessarily, be used in a steering segment.
Function

The #LOWER directive signifies to the compiler that the succeeding statements are data statements requiring storage space in the lower data area. (For PLAN 1, see Note 4 below.)

Format

#LOWER is written in the label field. The operation and accumulator fields are left blank. The first and following lines of the operand field may be used to specify the areas to be allocated lower data storage. These data statements may be either

1. variable data statements, or
2. constant data statements.

Notes

1. The rules for writing variable and constant data statements are given in Chapter 2 of this manual under the heading Formats of Data Statements (pages 8 et seq.). Instruction statements may be accepted as data statements under a #LOWER directive provided that they do not appear on the same line as the directive; where instruction statements are written under a #LOWER directive the label, operation and accumulator fields may of course be used as necessary (Chapter 2 pages 13 and 14).

2. #LOWER may appear any number of times in a source program with the following restrictions:
   (a) If a storage location is to be defined under a #LOWER directive, then the definition must precede the use of the name in the program.
   (b) Each #LOWER directive indicates a new section of lower memory that is not necessarily adjacent to that formed by the previous #LOWER directive.

3. Since the appearance of a symbol under this type of directive constitutes a definition of that symbol, a second appearance of the same symbol under this or any other defining directive would be ambiguous and would be treated as an error (unless the symbol appeared in an expression; see #DEFINE).

4. In PLAN 1, program instructions and data are implicitly held in lower memory and, therefore, the #LOWER directive causes no action during compilation. The #LOWER directive may be written as specified above with the exception of the following: the data statements must be written starting on the line after the #LOWER directive. Where variable and constant data statements are written under the same directive, each type of statement as in 'Format' above must be followed either by a statement of the same type or by a space, the rest of the line being left blank. The statements of the other type must then commence on the next line. See Chapter 2, page 3 if it is desired to use #LOWER in PLAN 1 in an upwards compatible manner. The PLAN 1 compiler will list #LOWER directives.

5. Data areas defined normally under a #LOWER directive can only be used by the segments in which they are allocated. If it is required to have several segments refer to the same area in lower data, the areas must be defined as follows:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>OPERAND</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>#LOWER</td>
<td>COMMON/BLKNAME1/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where BLKNAME1 is a name containing up to eleven alphanumeric characters of which the first must be alphabetic, and Blt refers to a list of data statements just as in the normal use of #LOWER. The effect is to set aside an area called BLKNAME1 in lower storage, and to make the area common to all segments in which a similar directive occurs. Each block name must be defined in a separate #LOWER directive, unique within the program and must not be the same as any segment name or cue name within the program. The items of the list must be unique within each segment. The block name must not be referenced in the operand field of a non-branch instruction.
If a #LOWER directive is written thus:

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>#LOWER</td>
<td>COMMON/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

the effect is to reserve an unnamed area in lower data storage, common to all segments in which a similar directive occurs. This area is identified in compiler listings and store analyses by the symbol %.

The facility is intended for use in mixed language programs, the area reserved being that which in a FORTRAN segment is the Blank Common Area. The directive would be followed by a list of data statements in the usual way.

In the case of overlay programs a use of the #LOWER directive is provided which enables common data areas to be provided for an overlay unit of a program only. Thus the common area will not be permanently present in store, but will only exist when the overlay unit to which it refers is present.

This is achieved by writing the #LOWER directive in the steering segment of an overlay program, in the following manner:

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>#LOWER</td>
<td>OVERCOMMON/ BLOCKNAME/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

No list follows the block name in the steering segment. The programmer would define his common data areas, as shown in the first of the above examples, in a normal segment of the program. The same block name that followed OVERCOMMON in the steering segment, however, would be given, and this would ensure that the data areas so defined would be common only to the segments within that unit of the overlay area.
Function

The #MACRO directive instructs the compiler that the statements which follow represent a user's macro-instructions.

It will often be found that a set pattern of PLAN statements must be written a number of times in a program, though with different operands, accumulators and modifiers. To save time, programmers may write these sequences of statements under the #MACRO directive with single-letter alphabetic parameters in place of the variable operands, accumulators and modifiers actually required in the program. When the program is compiled, this results in the generation of the sequence of statements defined under the #MACRO directive, with the alphabetic parameters replaced by the defined values.

Format

#MACRO is written in the label field, and the rest of this line is left blank. On the next line, the name assigned to the macro-instruction is written in the operation field, and the alphabetic parameters are written in the accumulator and/or operand fields. The series of instructions and/or data statements which constitute the macro-instruction are written on subsequent lines. Statements which constitute a macro-instruction may not have labels.

The name of the macro-instruction may consist of up to five alphanumeric characters, of which the first must be alphabetic. This name must be unique in the program. It cannot be the same as the name of any PLAN pseudo-operation (see Chapter 5), and it can only be the same as the name of a standard PLAN function if the number of accumulators differs from the number in the standard function.

Each alphabetic parameter must be unique within the macro-instruction, but may be used as desired elsewhere in the program. Parameters written in the operand field must be separated by commas; there is no comma after the last parameter. None, one or two of these parameters may also be placed in the accumulator field; there must be no commas in this field.

In operation statements which form part of the macro-instruction, only PLAN functions or PLAN macro-instructions may be used. PLAN pseudo-operations may not be used within a macro-instruction.

In data statements which form part of the macro-instruction, only one constant or one storage area may be defined on a line; further items separated by commas are not permitted.

Example

1) Consider this example of a user's macro-instruction named VERAD:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>#MACRO</td>
<td>VERAD</td>
<td>B,D,E,F,J,K,D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This macro-instruction may subsequently be used anywhere in the segment containing it, or in later segments, provided that such segments are not compiled separately. With each use of the macro-instruction the programmer must write its name and give actual values to the symbolic parameters. For example, to utilize the above macro-instruction and to give values to its parameters, an instruction like this might be written:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This instruction would result in

A being replaced by 3,
B by HELLO,
C by PATOD,
E by 2,
D by FRANK.
The above single-line instruction would then be equivalent to writing:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>VJX</td>
<td>3</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>AAQ</td>
<td>3</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>SFX</td>
<td>3</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>STD</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

During compilation, the macro-instruction would be expanded into this sequence of instructions, which would then be stored as part of the object program.

2 This example illustrates both constant and variable data statements used in a macro-instruction. The data storage would be allocated within the program area, in sequence with the program instructions, each time the macro-instruction is used.

| LABEL | OPERATION | ACC | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------|-----------|----|----|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #MACRO | CHECK     | 6  | 5  | 4  | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 9 |
|        |           | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 |
| #END   | CHECK     | 6  | 5  | 4  | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 9 | 8 |
|        |           | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 |
| BZ     | 7         | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 |
| BNZ    | 7         | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 |
| DISTY  | 7         | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 |

The macro-instruction would cause the message 'DISPLAY: TEST n FAIL' or the message 'DISPLAY: TEST n O.K.' to be output, depending on the contents of a specified accumulator. The data words generated by this particular macro-instruction would have to be below location 4096 of the program. The use of relative operands in non-branch instructions will cause H and C class errors to be flagged in the compiler listing of this macro-instruction (see Note 6).

Notes
1 Alphabetic parameters appearing in the accumulator fields of the instruction sequence must be given values of 0 to 7.
2 Parameters used as modifiers must be given values of 1 to 3.
3 Any use of a macro-instruction can be labelled. This is equivalent to labelling the first generated instruction or data word.
4 The last statement in a macro-instruction definition must be followed by a major directive.
5 A macro-instruction must be defined before it is used. The definition will be retained by the compiler until #FINISH is read, so that the macro-instruction may be used by other segments subsequent to the one in which it was defined. As the name of a macro-instruction must be unique within a program, it follows that it cannot be redefined in later segments of the program which are to be compiled in the same run; any attempt to do so will be flagged as a U-type error, and otherwise ignored by the compiler. If other segments using the macro are subsequently updated and recompiled individually, then the segment containing the definition of the macro must be recompiled in the same run, preceding the updated segment.
6 (a) As labels may not be used within a macro-instruction, it follows that if a branch is made from one instruction statement to another within a macro-instruction, then a relative operand of the form \( * \pm N \) must be used.

(b) Similarly, if an instruction statement within a macro-instruction references a constant which has been defined in the macro-instruction, it must do so by means of a relative operand.

(c) Care must be taken that any data areas, whether constant or variable, defined in a macro-instruction are generated below location 4096 of the program, unless all addressing of those areas is by means of modification.

(d) Statements discussed in section (b) of this note, and statements which address variable data areas allocated in a macro-instruction, will be flagged on compiler listings with H and C class error warnings. Provided that the restriction in section (c) of this note is observed, these error flags may be disregarded.

7 If a macro-instruction is to be used more than once in a program, any allocations of variable data storage made within it should be by means of alphabetic parameters (see Example 2); for if a macro-instruction contains a variable data statement having a symbolic name in the operand field and the macro-instruction is used more than once in a program, then the symbolic name is no longer unique within the program, and compilation errors will result.

8 It is possible to nest user's macro-instructions to a depth of 1, i.e. any user's macro may employ a previously-defined user's macro provided the latter does not itself contain any macro (whether user's macro or PLAN macro).

9 None of the three characters [, \( \uparrow \) and \( \pm \) should appear in data statements which specify characters ('nH' statements).
**#MONITOR**

**Function**
The #MONITOR directive instructs the compiler to call the Monitor package. This package prints out at run time the contents of certain areas of the store, in accordance with a list of parameters which follow the #MONITOR directive. This directive is used to provide information necessary for diagnosing program errors.

**Format**
- MONITOR is written in the label field, extending into the operation field. Parameters are written in the operand field and take the following format:
  
  $$\frac{n}{m}$$
  
  $$c/a$$

  where $n$ = an integer in the range 0 to 511, inclusive, that gives the number of lines in the following parameter list.

  $m$ = a three-digit number that identifies the point in the source program that occasions the output. If 1,000 is added to this number then no printing of the contents of the accumulators will take place.

  $c/a$ = a counter modifier, where $c$ is the number of words to be printed out, and $a$ is the address of the first word (which cannot be modified). $a$ must not be the address of an accumulator. The parameter $c/a$ will be repeated $n$ times.

The $c/a$ parameters are optional and, if they are omitted, the entry in the operand field of the directive takes the form $0/m$; this indicates that the contents of the accumulators only are to be printed out.

**Example**
At a certain point in a program it is required to print out:

1. The contents of a 35-word area beginning at the symbolic address PROFIT.
2. An area, whose size is determined by the absolute symbol NET, which starts at BALAN + 10.
3. The ten entry locations, words 20 to 29.

In the program this would be written:

| LABEL | OPERATION | ACC 1 | ACC 2 | ACC 3 | ACC 4 | ACC 5 | ACC 6 | ACC 7 | ACC 8 | ACC 9 | ACC 10 | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND | OPERAND |
|-------|-----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|---------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| #MONITOR | 3/14/31 |       |       |       |       |       |       |       |       |       |       |         | PROFIT   |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |
|         | 10/20    |       |       |       |       |       |       |       |       |       |       |         | BALAN+10 |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |

The resulting printout would include:

1. MONITOR PRINT 401
   (which identifies the source program #MONITOR point that occasions the output).
2. Contents of the accumulators, X0 to X7
   (which are always printed, except when 1,000 is added to the identifier $m$).
3. Contents of the areas specified in the parameter list (and defined at the beginning of this example).

**Notes**
1. MONITOR must be specified in the PLAN steering line, otherwise all #MONITOR directives and their associated parameter lists will be ignored by the compiler, which will not incorporate the Monitor package into the program.
When the Monitor package is first entered, it tests the bit of switch word 30 that corresponds to the first digit of identifier \( n \) (or the second digit if 1,000 is added to \( n \)). Monitor printing will not take place unless this bit is set to 1.

The Monitor package does not disturb any of the contents of either the accumulators or the user's data locations.

The settings of zero suppression mode and of \( V \) are preserved, though the original setting of \( C \) is lost.

Users of PLAN 1 and 2, though they may not use the \#MONITOR directive, may call the Monitor package as a subroutine.

Details of the Monitor package are given in Chapter 9 and in the PLAN Program Development Aids manual.

In a multi-member program, the Monitor package should only be called from one program member.

In PLAN 4, the routine MONITORX is called in place of the compact mode Monitor package if the \#P Mode directive specifies 22AM. The MONITORX routine will operate in any legal mode combination. It is fully described in the PLAN Program Development Aids manual.
Function

The #OMIT directive, given only in segmented programs, causes the compiler to regard as satisfied any cue which appears under the directive. This enables a program which has a segment missing for some legitimate reason to be consolidated without giving rise to exception conditions. It also enables a consolidator to ignore specified segments which may be present in its input stream.

Format

#OMIT is written in the label field. The operation and accumulator fields are left blank. In the first and following lines of the operand field are written the name(s) of the missing segment(s) and the cue names from the operand fields of any #CUE directives which the user wishes to be regarded as satisfied. Each name except the last on a line is followed by a comma.

Example

| LABEL | OPERATION | ACC | 01 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
|-------|-----------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| #OMIT |           | IEO |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        | SEG06, LABEL, REG7 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Notes

1. Once a segment has been specified under the control of an #OMIT directive it will not be available to the user program since any attempted branch to that segment will cause the program to be illegal.

2. A #OMIT directive may, but need not necessarily, be used in a steering segment.
**Function**

The `#ORDER` directive occurs only in the steering segment of an overlay program. It indicates to the compiler that the segments will be presented in the sequence required for consolidation. It is applicable only to tape compilers with integral consolidation.

**Format**

`#ORDER` is written in the label field. The operation, accumulator and operand fields are left blank.

**Notes**

1. If this directive is given the program must be presented to the compiler in the following manner:
   - Overlay areas in ascending sequence.
   - Overlay units within areas
   - Segments within overlay units in the sequence specified under the `#OVERLAY` directive.
   - Permanent program segments last.

   If this directive is given and the segments are not presented in the stipulated sequence compilation will fail.

2. If this directive is not given the compiler will assume that the segments are not in order, in which case compilation will take longer.

3. This directive cannot be used if library subroutines are included in the overlay units, by declaring them in the `#OVERLAY` directive. Library subroutines may be called by overlay segments, in which case the library subroutines will be placed in the permanent program.
Function

The `:OUST` directive deletes symbols from the symbol table when they are not required any more in the program. This prevents the table from becoming full, and frees the space taken up by each deleted symbol (two words) for new symbol definitions and branch labels.

Format

`:OUST` is written in the label field. The operation and accumulator fields are blank. The entry in the operand field takes the form

```
S1, S2, S3, S4, ...
```

where $S_1$, $S_2$, etc., represent a number of absolute, variable or preset symbols, of one to five characters each, that have already been defined (and not previously ousted) and are not to be referred to further down in the program.

The operand field entry may consist of as many symbols, separated by commas, as can fit in this field, and is terminated by the first space character.

Example

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>@</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>#OUST</td>
<td>RAVE, IS, TON, SO, DAB, HESS1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The above example would cause the compiler to delete the symbols RAVE, IS, TON, SO, DAB and HESS1 from its symbol table, and thus causes 12 words to become available for further symbol definitions or branch labels.

Notes

1. Once a symbol has been ousted, it cannot be referred to again in the program and any such reference will cause an error to be indicated. The symbol will not appear in the symbol list in the listing summary. The only exception to this is that a symbol may be redefined after having been ousted but this practice is inadvisable and likely to cause confusion.

2. If an `:OUST` directive appears in a program compiled by a PLAN 2, PLAN 3 or PLAN 4 compiler, an error will be given and the statement ignored. This will not generally affect the compiled result since there should not be any danger of the symbol table becoming full in this case.

3. Each line of symbols for deletion must be preceded by an `:OUST` directive.

4. If a line of symbols for deletion includes one that has not been previously defined, an operand error is indicated, but the compiler goes on to delete the other symbols in the line.
Function

The #OVERLAY directive occurs only in the steering segment of an overlay program. It indicates to the compiler which segments are to be part of an overlay unit, specifying an area and unit number for each overlay. Any segment not defined under #OVERLAY will be held in the permanent program area.

Format

#OVERLAY is written in the label field extending into the operation field. The entry in the operand field takes the form:

\[(A/u) \text{SIG}\]

where

A = area number (a decimal integer in the range 1 to 255),

u = unit number (a decimal integer in the range 1 to 1023),

SIG = segment name (a mixed alphanumeric field starting with an alphabetic character, to a maximum of eleven characters).

The entry in the operand field is repeated for each segment to be included in an overlay; segments mentioned in this list but not immediately preceded by an area and unit number will be considered to belong to the area last mentioned. Each entry, except the last on a line, is followed by a comma.

Example

| LABEL | OPERATION | SEG | | | | | OPERAND | | | | | |
|-------|-----------|-----|---|---|---|---|----------|---|---|---|---|---|---|
|       |           |     |   |   |   |   | (1/1) SEG1 | (1/1) SEG2 |   |   |   |   |
| #OVERLAY | (2/1) SEG3 | (2/2) SEG4 |   |   |   |   |   |   |   |   |   |
|        | (3/1) SEG5 | SEG6 |   |   |   |   |   |   |   |   |   |

Note

For further information on overlays, see Chapter 3, page 2, Overlay Programming; Chapter 5, pages 55 to 58, Overlay Macro-instructions; Chapter 6, page 32, #PERMANENT; and Appendix 6.
Function
The #PAGE directive will cause a throw to the next head of form before printing the next line of a compilation list.

Format
#PAGE is written in the label field, and the remainder of the line is left blank.

Notes
1 #PAGE is purely a directive for controlling the printed output of the compiler. It has no effect on the generated object program.
2 When a program is being compiled, the PLAN compiler is capable of listing the source and object programs on a line printer.
   When listing by a PLAN 3 or PLAN 4 compiler is taking place, the line printer normally throws to the head of the next form:
   (a) on commencing a new section of the list (i.e. a new segment, a segment's storage summary, 
       the consolidated program summary or a store map).
   (b) either on the expiry of a count of 60 lines or, if switch 18 is on, when punching is detected in
       channel 8 of the printer control loop.
   When listing by a PLAN 1 or PLAN 2 compiler is taking place, the line printer normally throws to 
   the head of the next form on the expiry of a count of 57 lines.
   The #PAGE directive can be used if it is desired to break up the listing of a segment into logical 
   subdivisions.
3 The #PAGE directive must only be used within a segment. If it is used before the #PROGRAM 
   directive defining the program/segment name, or following the #END directive, error conditions 
   will arise.
#PERIPHERAL

## Function

The #PERIPHERAL directive instructs the compiler which of the basic peripherals are to be allocated to the program.

## Format

#PERIPHERAL is written in the label field, extending into the operation field. The accumulator field is left blank. The symbolic names of the peripherals required by the program are written in the operand field, and may extend into the operand fields of the following lines. Each symbolic name except the last in a line is followed by a comma. The symbolic names that may be used for peripherals are as follows:

- Paper tape readers: TRn
- Paper tape punches: TPn
- Line printers: LPn
- Card readers: CRn
- Card punches: CPn

where n is a decimal integer in the range 0 to 6.

## Example

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>CR1</th>
<th>TP2</th>
<th>LP0</th>
</tr>
</thead>
<tbody>
<tr>
<td>#PERIPHERAL</td>
<td>CR1, TP2, LP0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

As explained in the Notes below, this coding would assign two card readers (CR0 and CR1), three paper tape punches (TP0, TP1 and TP2) and one line printer.

## Notes

1. Only the basic peripherals for which symbolic names are specified under ‘Format’ above may be assigned in this way.

2. Up to seven units of each type may be assigned by this directive. A program distinguishes between units of the same type by means of a relative unit number for each unit of that type (represented by n in the specification of the format above). For example, if a program uses three card readers they are referred to as CR0, CR1 and CR2.

3. If a #PERIPHERAL directive includes only one entry for a particular type of peripheral and its relative unit number is greater than zero, units of that type with all lower relative unit numbers are also assigned to the program.

4. A #PERIPHERAL directive may, but need not necessarily, appear in a steering segment.

5. If a program requires a particular model of any type of unit (e.g. a card reader capable of reading in binary mode), this cannot be specified by means of a #PERIPHERAL directive. It may be allocated instead either by operator action or, with dual program or multi-program Executives, by the ALLOT instruction, utilising the peripheral property code facility. The ALLOT instruction may in any case be used as an alternative method of allocating peripherals.
Function

The #PERMANENT directive is only used in segmented programs. Semicompiled segments listed under this directive will be consolidated, even if they are not called by segment name in another segment. Such segments may be entered by branches to cues (see #CUE, Chapter 6, page 7).

The primary function of the #PERMANENT directive is to ensure the consolidation of the appropriate run-time package into an overlay program. When used for this purpose, #PERMANENT must appear in the steering segment (see Chapter 3, page 2).

The various run-time packages for the different overlay media are held as subroutines in semicompiled form on the library. A library subroutine will normally be consolidated into the program only if it is called by its subroutine segment name (see Chapter 3, page 2, Subroutines). Each of the pseudo-operations BRING, ENTER and RECAL generates a branch to a cue which exists identically within every run-time package; no run-time overlay package is called by segment name. Therefore the appropriate package will only be consolidated if it is listed under #PERMANENT.

The user may also list under #PERMANENT any source segment which is to be held in permanent storage, but this is not necessary, as all source segments encountered by the compiler that are not listed under #OVERLAY (Chapter 6, page 29) will be in the permanent program area.

Format

#PERMANENT is written in the label field, extending into the operation field. The accumulator field is left blank. The entry in the operand field consists of the appropriate segment name(s). The name of the run-time overlay package, if present, must be the first of these. If there is more than one name then each name, except the last on a line, is followed by a comma.

The following run-time overlay packages are available. Each subroutine controls the bringing in of overlays from the specified medium.

%EROL for overlays held on E.D.S., Twin-EDS. or F.D.S.
%TROL for overlays held on magnetic tape,
%AATROL for overlays held on magnetic tape when a real time device is in use,
%FINROL \{ for overlays held on disc or in core,
%FINROLX \} for overlays held on a magnetic drum,
%DROL for overlays held on a magnetic drum,
%CROL for overlays held on cassette tape.

For further details of these packages, see Appendix 6.

Example

If the #PERMANENT directive were to appear in the steering segment of an overlay program it might be written thus:

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Accum</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>#PERMANENT</td>
<td>TROL, SEG8, SEG9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This means that the segments are to be read in from magnetic tape, the name of the run-time package being %TROL. SEG8 and SEG9 are also to be included in the permanent part of the program.
Function

The #LOWER directive signifies to the compiler that the succeeding statements are data statements requiring storage space in the lower pure data area.

Format

#LOWER is written in the label field. The operation and accumulator fields are left blank. The first and following lines of operation field may be used to specify the areas to be allocated lower pure data storage.

Notes

1. #LOWER is used in the same way as the #LOWER directive and lower pure common blocks can be defined.
2. The data statements defined by the #LOWER directive are those statements which remain unchanged during the running of the program. Thus instruction statements, presets, literals and any constant data statements may be acceptable.
Function

The \#PMODE directive occurs only in the steering segment of a PLAN 4 program. It provides a means of specifying the address mode and branch mode in which the object program is to operate after loading, and also the checks with regard to address mode and branch mode that the compiler is to perform on the various segments of the program.

Format

\#PMODE is written in the label field. The operation and accumulator fields are left blank. The entry in the operand field takes the form:

\[ nAM, nCH, nBM \]

where  \( nAM = \) either 15AM or 22AM. This specifies the address mode in which the object program is to operate after loading. (For an explanation of address modes see Chapter 1 page 6.) \( nAM \) may be omitted, in which case 15AM is assumed.

\( nCH = \) either 15CH, 22CH or MIXAM. This specifies what checks on address mode compatibility should be carried out on the segments comprising the program.

15CH Reject all segments that are not specified explicitly or implicitly on their first \#PROGRAM line as functioning correctly in 15-bit address mode.

22CH Reject all segments that are not specified explicitly on their first \#PROGRAM line as functioning correctly in 22-bit address mode.

MIXAM Perform no check on the address mode.

\( nCH \) may be omitted, in which case the check is carried out in accordance with the setting of the initial address mode, as determined by \( nAM \).

\( nBM = \) either DBM or EBM. This specifies the branch mode in which the object program is to operate after loading, and also what checks on branch mode compatibility are to be carried out on the segments comprising the program.

DBM Reject all segments that are not specified explicitly or implicitly on their first \#PROGRAM line as functioning correctly in direct branch mode.

EBM Reject all segments that are not specified explicitly on their first \#PROGRAM line as functioning correctly in extended branch mode.

\( nBM \) may be omitted, in which case DBM is assumed.

\( nAM, nCH \) and \( nBM \) may be in any order, separated by commas. If any is omitted, the redundant separating comma should also be omitted. If 15AM, 15CH, DBM is required, the operand field may be left blank.

Notes

1. A PLAN 4 program must have a steering segment, which must contain a \#PMODE directive. Only one \#PMODE directive is accepted in a program.

2. Checks on address mode and branch mode compatibility as specified by \( nCH \) and \( nBM \) are performed on each segment immediately after the compilation of the segment.

If LIST or SHORTLIST or FULLLIST is present in the steering line applicable to the segment, and the address mode compatibility check fails (i.e., if the first \#PROGRAM line of the segment does not indicate explicitly or, if applicable, implicitly that the segment is capable of functioning in an address mode acceptable to the \( nCH \) parameter) then the message

SEGMENT DATA MODE INCORRECT

is output following the error line in the segment listing.

If LIST or SHORTLIST is present on the steering line applicable to the segment, and the branch mode compatibility check fails (i.e., if the first \#PROGRAM line of the segment does not indicate explicitly or, if applicable, implicitly that the segment is capable of functioning in the branch mode acceptable to the
nBM parameter) then the message

SEGMENT BRANCH MODE INCORRECT

is output following the error line in the segment listing.

If both these messages are output, the branch mode message precedes the data mode message.

With the compiler #XPLN, if CONSOLIDATE or a facility which implies CONSOLIDATE is present in the steering line applicable to the segment, and the address mode compatibility check fails, the segment is not included in the consolidated program. A failure of the address mode compatibility check on the first (steering) segment, however, will not result in the rejection of this segment.

If unconsolidated semi-compiled output is requested (see Chapter 8) and a segment fails its address mode compatibility check, the output tape will still contain that segment, though it will not be possible to consolidate the segment into the program of which it is normally a part.

#XPLN does not implement the extended branch mode features of PLAN 4, so if a #PMODE directive with nBM present is detected by that compiler it is error flagged, but the nAM, nCH parameter are dealt with correctly.

With the compiler #XPLT, if OBJECT is present in the steering line applicable to the segment, the information as to whether or not a segment has passed its address mode and branch mode compatibility checks is included in the semi-compiled output, and any segment which has failed those checks is not included in the object program by the consolidator. A failure of the address mode or branch mode compatibility check on the first (steering) segment, however, will not result in the rejection of this segment.

3 The checks on address mode and branch mode compatibility specified by nCH,nBM are applied also to any library subroutines called by the program.

4 If monitor points are included in the program being compiled and MONITOR is present in the steering information then nAM also determines which monitor routine is called, thus:

15AM specified or nAM omitted: MONITOR called.

22AM specified: MONITORX called.

(See further Chapter 8, under the relevant compiler, in the sub-section entitled 'Steering Information'.)
Function

The #PROGRAM directive is used for the following purposes:
1. Naming a program.
2. Naming a program segment.
3. Naming a master or subprogram or program member (PLAN 3 and 4 only).
4. Specifying the operating mode in which a segment is intended to work correctly (PLAN 4 only).
5. Introducing a group of operation statements.
7. Designating a segment as an "error segment", that is, a segment which will be treated by the consolidator as satisfying all otherwise unsatisfied cues (PLAN 4 only).
8. Generating replacers in extended branch mode programs (PLAN 4 only).

Each of these is dealt with separately below.

1  NAMING A PROGRAM

The first statement of any program must be a #PROGRAM directive that specifies the program name, which will be used in communication with Executive.

Format

#PROGRAM is written in the label field, extending into the operation field. The accumulator field is left blank. The program name is written in the operand field, and should consist of four alphanumeric characters, of which the first must be alphabetic.

For dual or multiprogramming processors, the program name is followed immediately by the priority value assigned to the program. This priority value consists of two decimal integers in the range 01 to 99. For a Priority Member (member 5) a priority value of octal 77 is permitted; it should be written as the two characters ←←.

Example

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC 123</th>
<th>Z 24</th>
<th>Z 8</th>
<th>B</th>
<th>E</th>
<th>OPERAND</th>
<th>N 4</th>
<th>N 5</th>
<th>N 6</th>
<th>N 7</th>
<th>N 8</th>
<th>N 9</th>
<th>76 54 02 01</th>
</tr>
</thead>
<tbody>
<tr>
<td>#PROGRAM</td>
<td>VERAGA.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2  NAMING A PROGRAM SEGMENT

Each segment of any program must be headed by a #PROGRAM directive that specifies the program name and the segment name.

Format

#PROGRAM is written in the label field, extending into the operation field. The accumulator field is left blank. The program name and priority value (if any) are written in the operand field according to the rules in (1) above. The priority value must be the same for all segments of a program or subprogram.

The name and priority value are followed by a solidus and the segment name. The segment name must be unique in the whole program and may consist of up to eleven alphanumeric characters, of which the first must be alphabetic.
3  NAMING A MASTER OR SUBPROGRAM OR PROGRAM MEMBER

Each program segment which is (or is part of) a master or subprogram, or program member, must be headed by a 
#PROGRAM directive that specifies, as relevant, the master program number, subprogram number or member
number, as well as the program name and segment name.

Format

#PROGRAM is written in the label field, extending into the operation field. The accumulator field is left blank.

The first column of the operand field contains either 0, 1, 2, 3, or 5. 0, 1, 2 or 3 are subprogram or member
numbers. 5 may be used only in a program which includes a Priority Member, and indicates the Priority Member. The
second and following columns of the operand field contain the program name and the priority value assigned to the
subprogram or program member. Each priority value assigned must be unique. This entry is followed by a solidus
and the segment name (if applicable). The program name, segment name and priority value must accord with the
rules outlined in 1 and 2 above.

Example

| LABEL | OPERATION | ACC. | 15 | 2 | 24 | 26 | 2 | 32 | X | 40 | 44 | 48 | 52 | 56 | 60 | 64 | 68 | 72 | 76 | 80 |
|-------|-----------|------|----|---|----|----|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|
| #PROGRAM | lm2 | a | 76 | / | f | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

4  SPECIFYING THE OPERATING MODE IN WHICH A SEGMENT IS INTENDED TO WORK

Each segment in a PLAN 4 program must be headed by a 
#PROGRAM directive that specifies, in addition to the
program name and segment name, the operating mode in which the segment is intended to work correctly.

Format

#PROGRAM is written in the label field, extending into the operation field. The accumulator field is left blank.
The operand field is of the same format, and the items comprising it must conform to the same rules, as described
for 2 or 3 above or 6 or 7 below, with the addition that address mode and branch mode information follows the
segment name. The address mode and branch mode information comprises any combination of the following
items, separated by commas, enclosed together in parentheses:

15AM  The segment is intended to work correctly if entered in 15-bit address mode.
22AM  The segment is intended to work correctly if entered in 22-bit address mode.
DBM   The segment is intended to work correctly if consolidated into a program whose overall branch mode
        is direct branch mode.
EBM   The segment is intended to work correctly if consolidated into a program whose overall branch mode is
        extended branch mode.

If address mode information is omitted from the coding, the 15AM specification is implied.
If branch mode information is omitted from the coding, the DBM specification is implied.

Examples

| LABEL | OPERATION | ACC. | 15 | 2 | 24 | 26 | 2 | 32 | X | 40 | 44 | 48 | 52 | 56 | 60 | 64 | 68 | 72 | 76 | 80 |
|-------|-----------|------|----|---|----|----|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|
| #PROGRAM | jms | e | 76 | / | p | 72 | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 76 |

4322(7.72)  Chapter 6  37
The second example specifies that the segment is intended to work correctly irrespective of the address mode in which it is entered (either because it avoids features specific to one address mode, or because it switches the address mode if necessary) but that it may not be consolidated into an extended branch mode program.

In addition, in a paged environment the purity of the segment may be defined (#XPLT only). The following parameters are available:

- **PURE** The segment is intended to work correctly as a shareable segment, that is, the operation statements defined under a #PROGRAM directive (see below) may be shared.
- **IMPURE** The segment is intended to work correctly as a non-shareable segment, that is, the operation statements defined under a #PROGRAM directive (see below) will not be shared.

If the purity of the segment is not defined, the setting of bit 15 of the #XPLT switch word will control the parity of the segments operation statements (see Chapter 8).

### Example

```
LABEL  OPERATION  ACC.  NO.   NO.   NO.   NO.   NO.   NO.   NO.   NO.
#PROGRAM  $ROBUT 60  WR  ITE  SEG (15AM, 22AM)
```

In the example the segment is intended to work correctly as a pure segment and as such will be marked by the system as shareable.

## 5 INTRODUCING A GROUP OF OPERATION STATEMENTS

Each group of operation statements in a PLAN 4, PLAN 3 or PLAN 2 program must be preceded by a #PROGRAM directive. This instructs the compiler to store the subsequent information in the program operation store.

As explained in Chapter 2, this is not applicable in PLAN 1. A #PROGRAM directive used in this context causes no action during compilation of a PLAN 1 program, but it should be used if upwards compatibility with PLAN 4, PLAN 3 or PLAN 2 is required.

Data statements may appear among the operation statements following a #PROGRAM directive. If constant data statements appear they will be compiled in the same way as if they had appeared under a #LOWER or #UPPER directive, except that the resulting presets will be stored in sequence with the accompanying program instructions in the program operation store, instead of in the lower or upper data store. Variable data statements may be used under a #PROGRAM directive in PLAN 4 and PLAN 3 only. Storage will then be allocated within the program operation store; the initial contents of this storage are undefined.

### Format

#PROGRAM is written in the label field, extending into the operation field. The rest of the line must be blank.

## 6 NAMING A FREE-STANDING SEGMENT

A free-standing segment must be headed by a #PROGRAM directive that specifies the segment name.

### Format

#PROGRAM is written in the label field, extending into the operation field. The accumulator field is left blank. A solidus is written in the operand field followed by the segment name. The segment name must be unique within any program incorporating the segment, and may consist of up to eleven alphanumeric characters, of which the first must be alphabetic. The segment name may, in PLAN 4 only, be followed by address mode, branch mode and purity information (see 4 above) and/or "error segment" designation (see 7 below).
Example

Note that no program name is given. This is particularly important where the segment is to be included in a subroutine block on a library tape. If in these circumstances a program name were present for the segment, then the segment would be incorporated into every program for which the consolidator searched the subroutine block, whether the segment was called by the program or not.

7 DESIGNATING A SEGMENT AS AN “ERROR SEGMENT”

In PLAN 4, a segment that is to be treated by the consolidator as satisfying all otherwise unsatisfied cues in a program must be headed by a #PROGRAM directive that specifies the segment name and designates it as an “error segment”.

Format

#PROGRAM is written in the label field, extending into the operation field. The accumulator field is left blank. The operand field is of the same format, and the items comprising it must conform to the same rules, as described for 2, 3, 4 or 6 above, with the addition that the two characters .E must follow the segment name. The address mode, branch mode and purity information, if present, may either follow .E or be placed between these characters and the segment name.

Examples

Example 1 illustrates the designation of an “error segment” to be called from a subroutine block on a library tape. No program name is given, for the reasons indicated under 6 above. Any program which requires it to be called for consolidation will specify it by means of a #ERRORSEG directive (see page 15).

Example 2 illustrates the designation of an “error segment” to be included in a source stream. If it is included in the source stream no #ERRORSEG directive is required in the program.

8 GENERATING REPLACERS IN EXTENDED BRANCH MODE PROGRAMS

In PLAN 4, a #PROGRAM directive which introduces a group of operation statements (see 5 above) may be used to generate extra replacers. Replacers are described in Chapter 1, page 8, and the generation of them is discussed in Chapter 8, page 32.

Format

#PROGRAM is written in the label field, extending into the operation field. The accumulator field is left blank. An absolute expression (for example, a decimal integer, an octal integer or a previously defined symbol) with a value in the range 0 to 510 is written in the operand field. See under #DEFINE for the rules for writing absolute expressions.

The absolute expression in the operand field of the #PROGRAM directive specifies the number of consecutive locations, commencing with the second location of the segment’s program operation store, for which replacers are to be generated. Such replacers are stored consecutively with the one generated for the first location of the segment’s program operation store, which is automatically provided when the segment is compiled to operate in extended branch mode.
If the segment is consolidated into a direct branch mode program, the specification contained in any absolute expression in the operand field of the #PROGRAM directive is ignored.

The absolute expression specifying the generation of additional replacers may be placed in the operand field of any #PROGRAM directive within the segment, other than the one naming the segment. If more than one #PROGRAM directive within a segment contains such an expression in its operand field, the first one encountered is accepted, and the operand fields of the others are ignored, except that they cause the statements to be flagged as O class errors.

If the absolute expression has a negative value or a value greater than 510, the statement is flagged as a J class error and 510 replacers are generated.
Function
The #PUPPER directive signifies to the compiler that the succeeding statements are data statements requiring storage space in the upper pure data area.

Format
#PUPPER is written in the label field. The operation and accumulator fields are left blank. The first and following lines of the operation field may be used to specify the areas to be allocated to upper pure data storage.

Notes
1. #PUPPER is used in the same way as the #UPPER directive and upper pure common blocks can be defined.
2. The data statements defined by the #PUPPER directive are those statements which remain unchanged during the running of a program. Thus, instruction statements, presets, literals and any constant data statements may be acceptable.
#SET

(Minor) PLAN 2,3,4

Function

The #SET directive instructs the compiler that the group of characters appearing before the equals sign in the operand field is to be assigned a value equal to that of the expression appearing after the equals sign. This directive is similar to the #DEFINE directive. The main difference is that a symbol defined by a #SET directive may subsequently be redefined by another #SET directive, provided that each new definition places the symbol in the same relative area. It may not, however, be defined in any other manner. The value given to the symbol on its appearance in any program statement will be that assigned by the nearest previous #SET directive.

Format

#SET is written in the label field. The operation and accumulator fields are left blank.

The parameters in the operand field take the form:

```
SYMBOL = EXPRESSION
```

Example

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>OPER</th>
<th>OPRT</th>
<th>OPRT</th>
<th>OPRT</th>
<th>OPRT</th>
<th>OPRT</th>
<th>OPRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>@SET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@LDN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@STO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@SET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@LDN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@STO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

In this example, LDN 4 MAX will be compiled in the first case as LDN 4 100, and in the second case as LDN 4 200.

Notes

1. A symbol may be reset relative to its own previous setting; for example:

```
<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>OPER</th>
<th>OPRT</th>
<th>OPRT</th>
<th>OPRT</th>
<th>OPRT</th>
<th>OPRT</th>
<th>OPRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>@SET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@LDN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@SET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@LDN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

2. In a compilation list, under the AB heading (see page 4 of Chapter 9), only the final values of SET symbols are printed. They are preceded by “S”.

3. The symbol may not be used in the accumulator field, nor to specify the modifier.

4. The symbol may not be used as a parameter for a pseudo-operation.

5. The final value of any expression must be in the range 0 to +32,767 (during evaluation, a full 24-bit word is used).
Function

The #STOP directive informs the compiler that there is no more input for this run. The compiler releases all peripherals, except for the program library tape or disc file from which it was loaded if the compiler is an overlay program, and halts. If a compiler parameter specified a successor program (see Chapter 7), then instead of halting the compiler deletes itself with a FIND message to initiate the loading of the successor program.

Format

#STOP is written in the label field. The operation, accumulator and operand fields are left blank.

Notes

1 The #STOP directive is applicable only to PLAN 3 batch compilers with basic peripheral input. (A parameter of similar formats is, however, used with COSY and DISC COSY editor programs, as described in Chapters 10 and 11 respectively.)
Function
The #SWITCH directive causes the compiler to change its basic peripheral input medium. If the present input device is a paper tape reader, it is released and a card reader is allocated. If the present input device is a card reader, it is released and a paper tape reader is allocated.

Format
#SWITCH is written in the label field, extending into the operation field. The rest of the line is left blank.

Notes
1  The #SWITCH directive is applicable only to PLAN 3 and PLAN 2 compilers with basic peripheral input. (A similar directive is, however, used with COSY and DISC COSY editor programs, as described in Chapters 10 and 11 respectively.)
2  The #SWITCH directive must not be used while compiler parameters are being read. It may be used immediately before the first steering line, or at any time thereafter.
3  The #SWITCH directive is not available to change the source input medium in PLAN 4, but a parameter of similar format may be used with #XPLN to change the input medium of the parameter stream. See Chapter 8.
Function

The #UPPER directive signifies to the compiler that the succeeding statements are data statements requiring storage space in the upper data area. (For PLAN 1, see Note 4 below.)

Format

#UPPER is written in the label field. The operation and accumulator fields are left blank. The first and following lines of the operand field may be used to specify the areas to be allocated upper data storage. These data statements may be either

1. variable data statements, or
2. constant data statements.

Notes

1. The rules for writing variable and constant data statements are given in Chapter 2 of this manual under the heading Formats of Data Statements (pages 8 et seq.). Instruction statements may be accepted as data statements under a #UPPER directive provided that they do not appear on the same line as the directive: where instruction statements are written under a #UPPER directive the label, operation and accumulator fields may of course be used as necessary (Chapter 2 pages 13 and 14).

2. #UPPER may appear any number of times in a source program with the following restrictions:

   (a) If a storage location is to be defined under a #UPPER directive, then the definition must precede the use of the name in the program.

   (b) Each #UPPER directive indicates a new section of upper memory that is not necessarily adjacent to that formed by the previous #UPPER directive.

3. Since the appearance of a symbol under this type of directive constitutes a definition of that symbol, a second appearance of the same symbol under this or any other defining directive would be ambiguous and would be treated as an error (unless the symbol appeared in an expression; see #DEFINE).

4. In PLAN 1, program instructions and data are implicitly held in lower memory (see Chapter 2, page 3) and, therefore, the #UPPER directive causes no action during compilation. The #UPPER directive may be written as specified above with the exception of the following: the variable or constant data statements must be written starting on the line following the #UPPER directive. Where variable and constant data statements are written under the same directive, each type of statement must be followed either by a statement of the same type or by a space, the rest of the line being left blank. The statements of the other type must then commence on the next line.

The PLAN 1 compiler will list #UPPER directives.

5. Locations in the upper data area cannot be directly addressed because they are likely to lie above the absolute location 4095. As explained in Chapter 3, an unmodified instruction can address only the first 4096 locations. The only method of utilizing information stored in the upper data area is by indirect addressing using a modification register.

For instance, a word PRICE is stored in the upper data area. It is required to load the contents of this word into accumulator 1:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>44</th>
<th>48</th>
<th>52</th>
<th>56</th>
<th>60</th>
<th>64</th>
<th>68</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4322(7.72) Chapter 6 45
A constant equal to the absolute address of PRICE is stored in location N in the lower data area, and the first instruction is obeyed as if it read:

LDX 2 N

That is, the value of the absolute address of PRICE is loaded into accumulator 2.

The second instruction results in the contents of the word whose address is

0 + contents of X2

being loaded into accumulator 1. That is, the contents of PRICE are loaded into X1.

Data areas defined normally under a #UPPER directive can only be used by the segments in which they are allocated. If it is required to have several segments refer to the same area in upper data, the areas must be defined as follows:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>48</th>
<th>56</th>
<th>64</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>#UPPER</td>
<td>COMMON/BLOCKNAME1</td>
<td>list</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where BLOCKNAME1 is a name containing up to eleven alphanumeric characters of which the first must be alphabetic, and list refers to a list of data statements just as in the normal use of #UPPER. The effect is to set aside an area called BLOCKNAME1 in upper storage, and to make the area common to all segments in which a similar directive occurs. Each block name must be defined in a separate #UPPER directive, must be unique within the program and must not be the same as any segment name or cue name within the program. The items of the list must be unique within each segment. The block name must not be referenced in the operand field of a non-branch instruction.

If a #UPPER directive is written thus:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>48</th>
<th>56</th>
<th>64</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>#UPPER</td>
<td>COMMON/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the effect is to reserve an unnamed area in upper common data storage, common to all segments in which a similar directive occurs. This area is identified in compiler listings and store analyses by the symbol % The facility is intended for use in mixed language programs, the area reserved being that which in a FORTRAN segment is the Blank Common Area. The directive would be followed by a list of data statements in the usual way.

In the case of overlay programs a use of the #UPPER directive is provided which enables common data areas to be provided for an overlay unit of a program only. Thus the common area will not be permanently present in store, but will only exist when the overlay unit to which it refers is present. This is achieved by writing the #UPPER directive in the steering segment of an overlay program, in the following manner:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>ACC</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>48</th>
<th>56</th>
<th>64</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>#UPPER</td>
<td>OVERCOMMON/BLOCKNAME1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No list follows the block name in the steering segment. The programmer would define his common data areas, as shown in the first of the above examples, in a normal segment of the program. The same block name that followed OVERCOMMON in the steering segment, however, would be given, and this would ensure that the data areas so defined would be common only to the segments within that unit of the overlay area.