IBM

# Programming Systems Analysis Guide

# IBM 7030 STRAP II

# PREFACE

Machine Oriented Programming has written this manual describing the functions of the STRAP II (STRETCH Assembly Program) in order to aid technical personnel in gaining a better understanding of the program.

Since initial release of the program, modifications have been made; these should not affect the technical accuracy at this level of documentation.

Certain knowledge concerning the programming system is a prerequisite for full utilization of this manual. Such background can be obtained from the IBM 7030 Reference Manual, Form A22-6530-2, 7030 STRAP II Reference Manual, Form C28-6129, STRAP II Assembly Listing, and IBM Master Control Program, Form C22-6678.

The reader must note that this analysis guide does not offer a complete technical picture of STRAP II, but does furnish valuable background and a general look at some of the more important aspects of the program.

Copies of this and other IBM publications can be obtained through IBM Branch Offices
Address comments concerning the content of this publication to:
IBM Corporation, Department B71, PO Box 390, Poughkeepsie, N. Y.

GENERAL INTRODUCTION . . . . . 7

INPUT-OUTPUT . . . . . 8
Communication Record Specifications . . . . 8
Format of STRAP II's Output on Disk . . . .10

DESCRIPTION OF THE ASSEMBLY PHASES .12
Conventions Adopted in the Descriptions of the Phases and of the Subroutines . . . .12
Introduction to the Phase Action in STRAP II. .12
Description of the Action Indicated on the Flow Chart of Pass 1 . . . . . .19
Description of the Action Indicated on the Flow Chart of Pass 2 . . . . . .32

DESCRIPTION OF THE MAJOR LOGIC AREAS WITHIN THE PHASES . . . . . .37
Major Logic Areas in the Common Section of STRAP II . . . . . .37
Subroutines--ERRIN, ERR, ERRNUM, and ERRPRT . . . . . .37
MOVE . . . . . .41
TABMAN . . . . . .44
SEARCH . . . . . .47
ADDORD . . . . . .48
ADDUNO . . . . . .49
ANEXTX . . . . . .51
RSERCH . . . . . .53
VALUE . . . . . .55
XERR . . . . . .62
Major Logic Areas in the Pass 1 Phase of STRAP II . . . . . .63
BOPSER . . . . . .67
CCA8 . . . . . .69
CPLTNM . . . . . .71
CPLTSY . . . . . .74
GETCHA . . . . . .79
GETFLD . . . . . .83
GNLOAD . . . . . .88
INTOUT . . . . . .90
MBSPEC . . . . . .93
MIBA . . . . . .95
MIOD . . . . . .97
MQDD . . . . . 100
MQDALF . . . . . 102
MQDNUM . . . . . 107
DNUM . . . . . 110
MQDDPA . . . . . 112
MQP . . . . . 115
MQR . . . . . 117
MDIMRF (and MDIMRT) . . . . . 119
MQS . . . . . 122
MQTAIL and MQT . . . . . 125
MULTI . . . . . 127
MX . . . . . 129
XINIT . . . . . 131
XINPUT . . . . . 133
Major Logic Area in the Between-Phases of STRAP II . . . . . 135
UITER . . . . . 135
Major Logic Areas in the Pass 2 Phases of STRAP II . . . . . 138
DECODE . . . . . 140
INTIN . . . . . 143
INSERT . . . . . 144
NEXT . . . . . 146
NMCP . . . . . 148
OUTPUT . . . . . 157
NAMEIN . . . . . 161
OPPUN . . . . . 163
OEDIT . . . . . 165
OPLIST . . . . . 167
NQBTP . . . . . 169
NUNDSY . . . . . 172

SERVICEABILITY AIDS . . . . . 173
Error Detection and Post Mortem Dumping . 173
Determination of Phase, Unit, and Symbol Table Entry Being Processed . . . . . 173
Locations in STRAP II Containing Information Used to Analyze Error Conditions . . . . 173
List of the STRAP II Numbered Error Messages . . . . . 176
List of the STRAP II I-O Error Messages . 179

INCLUDING A COPROCESSOR SUBROUTINE WITH STRAP II . . . . . 180

RELOCATION PSEUDO OPS . . . . . 183
An Example of a STRAP II Program Using Relocation Pseudo Ops . . . . . 184
Some Changes in the STRAP II Formats . . 184
Resume of the Processing of Relocation Pseudo Ops in STRAP II . . . . . 186
Formats of the Relocatable Binary Output Produced by OUTPUT . . . . . 187

APPENDIX A - FORMATS . . . . . 189

APPENDIX B - ADDITIONAL INSTRUCTIONS AND PSEUDO OPS ACCEPTED BY STRAP II 195

APPENDIX C - CODED EXPRESSIONS . . 197

APPENDIX D - SYMBOLIC DESIGNATIONS IN STRAP II . . . . . 202

APPENDIX E - TERMS . . . . . 203

APPENDIX F - PROGRAMMING WITH STRAP II . . . . . 207

APPENDIX G - COMPOSITION OF STRAP II's BINARY DECK . . . . . 208

APPENDIX H - MEMORY MAP OF STRAP II 209

LIST OF CHARTS


Chart AA. Main Pass 1 - Page 1 of 5 14
Chart AB. Main Pass 1 - Page 2 of 5 15
Chart AC. Main Pass 1 - Page 3 of 5 16
Chart AD. Main Pass 1 - Page 4 of 5 17
Chart AE. Main Pass 1 - Page 5 of 5 18
Chart FA. Pass 2 - Page 1 of 4 28
Chart FB. Pass 2 - Page 2 of 4 29
Chart FC. Pass 2 - Page 3 of 4 30
Chart FD. Pass 2 - Page 4 of 4 31
Chart RF. MOVE 40
Chart RA. TABMAN 43
Chart RB. ANEXT 50
Chart RW. RSERCH 52
Chart RD. VALUE 54
Chart WA. XERR (Disk Unit Check Fix-Up) 61
Chart BA. BOPSER - Page 1 of 2 65
Chart BB. BOPSER - Page 2 of 2 66
Chart BC. CCA8 68
Chart BD. CPLTNM 70
Chart BE. CPLTSY - Page 1 of 2 72
Chart BF. CPLTSY - Page 2 of 2 73
Chart BG. GETCHA - Page 1 of 2 77
Chart BH. GETCHA - Page 2 of 2 78
Chart BI. GETFLD - Page 1 of 3 80
Chart BS. GETFLD - Page 2 of 3 81
Chart BK. GETFLD - Page 3 of 3 82
Chart GN. GNLOAD 87
Chart IO. INTOUT 89
Chart BL. MBSPEC 92
Chart CA. MIBA 94
Chart CB. MIOD 96
Chart CS. MQDD - Page 1 of 2 98
Chart CK. MQOD - Page 2 of 2 99
Chart CC. MQDALF 101
Chart CF. MQDNUM - Page 1 of 3 104
Chart CG. MQDNUM - Page 2 of 3 105
Chart CH. MQDNUM - Page 3 of 3 106
Chart CD. DNUM - Page 1 of 2 108
Chart CE. DNUM - Page 2 of 2 109
Chart CI. MQDDPA 111
Chart DA. MQP 114
Chart DB. MQR 116
Chart DD. MDIMRF and MDIMRT 118
Chart DE. MQS 121
Chart DF. TAILOR (MQTAIL) - Page 1 of 2 123
Chart DG. TAILOR (MQT) - Page 2 of 2 124
Chart RE. MULTI 126
Chart DI. MX 128
Chart DJ. XINIT 130
Chart XI. XINPUT 132
Chart GB. UITER 136
Chart DC. DECODE 139
Chart ID. INTIN - Page 1 of 2 141
Chart IJ. INTIN (Interrupts) - Page 2 of 2 142
Chart RC. NEXT 145
Chart EA. NMCP 147
Chart OR. OUTPUT - page 1 of 8 149
Chart OP. OUTPUT - Page 2 of 8 150
Chart OQ. OUTPUT (Pseudo-Ops) - Page 3 of 8 151
Chart OR. OUTPUT (Pseudo-Ops) - Page 4 of 8 152
Chart OS. OUTPUT (Pseudo-Ops) - Page 5 of 8 153
Chart OT. OUTPUT (Pseudo-Ops) - Page 6 of 8 154
Chart OU. OUTPUT (DR) - Page 7 of 8 155
Chart OV. OUTPUT (DD) - Page 8 of 8 156
Chart NI. NAMEIN 160
Chart OM. OPPUN 162
Chart EI. OEDIT 164
Chart OL. OPLIST 166
Chart EB. NPNSYM AND NQBTP 168
Chart EG. NUNDSY - Page 1 of 2 170
Chart EH. NUNDSY - Page 2 of 2 171

LIST OF ILLUSTRATIONS


Figure 1. I-O and Data Flow in the IBM 7030 Processing System Compiling Chain 7

Figure 2. STRAP II's Output on Disk 11

Figure 3. Error Message Record for Pass 1, Pass 2a, and Pass 2b 38

Figure 4. Error Message Record for UITER 38

Figure 5. Initial Error Message Record 38

Figure 6. Final Error Message Record for ERRPRT 39

Figure 7. Format of MOVE's Storage Words 42

Figure 8. Format for a Table Control Block 44

Figure 9. Format of VALUE's Output (without relocation fields) 58

Figure 10. Format of VALUE's Internal Tables 60

Figure 11. Format of a Control Block for BOPSER 67

Figure 12. Format of CPLTNM's Output in $L 71

Figure 13. Format of CPLTSY's Output in $R 75

Figure 14. Format of CPLTSY's $ Output 76

Figure 15. Format of the First Word in a Unit 90

Figure 16. Algorithm for Division 112

Figure 17. Format of the Last Word in the Fixed Portion of a Unit 184

Figure 18. Format of the Third Word in VALUE's Output 185

Figure 19. Format of the Fifth Word in VALUE's Heading 186

Figure 20. Format of the Fixed Portion of an Intermediate Expanded Instruction Unit (without relocation fields) 189

Figure 21. Format of the Argument Portion of each of the Tables Handled by Table Management 190

Figure 22. Entries in the Variable Length Entry (vle) Table 191

Figure 23. Entries in the Table of Primary Operations 192

Figure 24. Entry in the Table of Secondary Operations 192

Figure 25. Entry in the Table of System Symbols 193

Figure 26. Entry Made in the Variable Portion of a Numeric DD's Expanded Instruction Unit 193

Figure 27. Dimension Entry 194

Figure 28. Coded Expression - Example 1 200

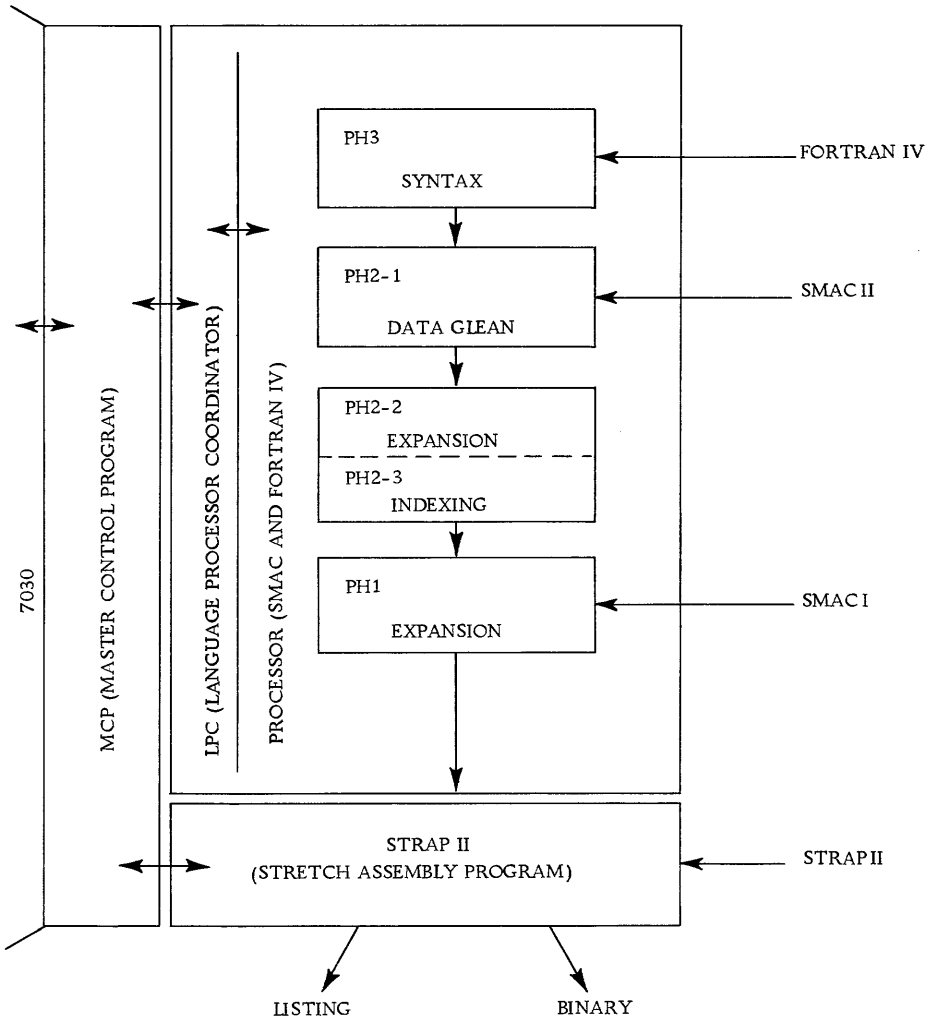
Figure 29. Coded Expression - Example 2 201

Figure 30. Memory Map of STRAP II 209

The IBM 7030 Assembly Program (STRAP II) is that program in the STRETCH programming system compiling chain which accepts as input, symbolic instructions of the IBM 7030 instruction set, and produces as output, IBM 7030 binary output and an assembly listing. The listing includes not only the representation of the symbolic input with the assigned binary output and location counter values, but also lists of error messages, unused symbols, undefined symbols, circularly defined symbols, and multiply defined symbols.

Since the minimum IBM STRETCH compiling system consists of the IBM 7030 Master Control Program (MCP) and STRAP II, the system requirements for STRAP II are the same as those of MCP.

Before describing the input-output specifications for STRAP II, it is necessary to establish the logical position of STRAP II in the compiling system. STRAP II's environment is set forth by the companion diagram where boxes represent programs, double-headed arrows represent I-O, and single headed arrows represent data paths for the requested levels.




| FORTRAN IV | SMAC II | SMAC I | STRAP | – | REQUEST LEVEL (DETERMINED BY MCP) |
|---|---|---|---|---|---|
| LPC | LPC | PPC | | | |
| STRAP | STRAP | STRAP | STRAP | | MCP CHAIN |
| LPC | LPC | LPC | | | (OF PROGRAMS BETWEEN |
| BSS | | | | | FIRST AND FINAL USE OF MCP) |

Figure 1. I-O Flow and Data Flow in the IBM 7030 Processing System Compiling Chain

## COMMUNICATION RECORD SPECIFICATIONS

A common Communication Record (COMREC) is located in core storage. It is used by the Master Control Program (MCP) to communicate with the processors, by the processors to communicate with each other, and by the processors to communicate with MCP. The first eight and a half words of the 25 word table are for the exclusive use of MCP. The use of the remainder of the COMREC is determined by the requirements of the compiling chain.

STRAP II expects words 7.32 through 24.63 in the COMREC to be set up as indicated in the following tables. The numbers in the tables referring to the code and to the unit are defined as follows:

| Code | | Unit | |
|---|---|---|---|
| 0 | Not used | 0 | System Input or System Output |
| 1 | Card code | 1-10 | Not used |
| 2 | A6 code | 11 | Disk |
| 3 | A8 code | | |
| 4 | Not used | | |
| 5 | IQS code | | |
| 6 | Binary code | | |
| 7 | Not used | | |

### COMMUNICATION RECORD

| Bits | Purpose | |
|---|---|---|
| 0.00 - 7.31<br>7.32 - 7.63 | Reserved for MCP | - - - - - |
| 7.32 - 7.39 | - - - - - | - - - - - |
| 7.40 | A pre-processor can turn on this bit. | LAST IN A CHAIN:<br>If this bit is on when STRAP II is last in a chain, STRAP II turns on MCP's reject bit. |
| 7.41 | STRAP II turns on this bit if STRAP II has determined the existence of one or more undefined, multiply-defined, and/or circularly defined symbols. | LAST IN A CHAIN:<br>In the case of -- COMPILGO, STRAP -- MCP's reject bit will be turned on if Bit 7.41 and/or Bit 7.42 has been turned on.<br><br>OTHER THAN LAST IN CHAIN:<br>These bits are available for examination by the post-processor. |
| 7.42 | STRAP II turns on this bit if STRAP II has encountered any of the following conditions:<br>Improper operation code<br>Improper secondary operation code<br>Entry mode preceding a non-DD<br>Extra secondary operation code<br>More than one DDS<br>Location counter does not agree<br>Assembly error<br>Symbolic statement too long | |
| 7.43 | STRAP II turns on this bit if any error condition which has not been suppressed has occurred. | |
| 7.44 - 7.63 | | |

## Input File Data to STRAP II

| Bits | Purpose |
|---|---|
| 8.00 - 8.63 | (This word must always be set up by the pre-processor.) |
| 8.00 - 8.17 | Relative arc number. |
| 8.18 - 8.24 | - - - - - |
| 8.25 - 8.27 | Number which defines code used. |
| 8.28 - 8.31 | Number which defines unit used. |
| 8.32 - 8.47 | - - - - - |
| 8.48 - 8.63 | Number of unit records to be written or read during a single I-O command. |

FIRST IN A CHAIN:
Input will be obtained from the System Input.

OTHER THAN FIRST IN A CHAIN:
Input will be from disk, or System Input, as specified by word 8.

## Listing Output From STRAP II

| Bits | Purpose |
|---|---|
| 9.00 - 9.63 | (This word must always be set up by the pre-processor.) |
| 9.00 - 9.17 | Relative arc number. |
| 9.18 - 9.24 | - - - - - |
| 9.25 - 9.27 | Number which defines code to be used. |
| 9.28 - 9.31 | Number which identifies unit to be used. |
| 9.32 - 9.39 | Number of lines to be placed on a page of listing output. |
| 9.40 - 9.47 | - - - - - |
| 9.48 - 9.63 | Number of unit records to be written or read during a single I-O command. |

LAST IN A CHAIN:
Output will be written via the System Output.

OTHER THAN LAST IN A CHAIN:
Listing will be written on the disk beginning in the relative arc specified in COMREC, or the System Output, as specified by word 9.

## Binary Output Data From STRAP II

| Bits | Purpose |
|---|---|
| 10.00 - 10.63 | (This word must always be set up by the pre-processor.) |
| 10.00 - 10.17 | Relative arc number. |
| 10.18 - 10.24 | - - - - - |
| 10.25 - 10.27 | Number which defines code to be used. |
| 10.28 - 10.31 | Number which defines unit to be used. |
| 10.32 - 10.47 | - - - - - |
| 10.48 - 10.63 | Number of unit records to be written or read during a single I-O command. |

LAST IN A CHAIN: COMPILE & GO:
Binary output will be written on disk beginning in relative arc 0.
If punch option exists in COMREC output will also be written via the System Output.

LAST IN A CHAIN: COMPILE ONLY:
Binary output will be written via the System Output.

NOT LAST IN A CHAIN:
Binary output will be written on the disk and the System Output.

STRAP II's Intermediate File 1 Data

| Bits | Purpose | |
|---|---|---|
| 11.00 - 11.63 | (STRAP II only uses a half of this word.) | |
| 11.00 - 11.17 | Relative arc number. | This file always goes on the disk. |
| 11.18 - 11.63 | - - - - - | |

STRAP II's Intermediate File 2 Data

| Bits | Purpose | |
|---|---|---|
| 12.00 - 12.63 | (STRAP II only uses a half of this word.) | |
| 12.00 - 12.17 | Relative arc number. | This file always goes on the disk. |
| 12.18 - 12.63 | - - - - - | |

STRAP II's Intermediate Name File

| Bits | Purpose | |
|---|---|---|
| 13.00 - 13.63 | (STRAP II only uses half of this word.) | |
| 13.00 - 13.17 | Relative arc number. | This file always goes on the disk. |
| 13.18 - 13.63 | - - - - - | |
| 14.00 - 24.63 | - - - - - | |

FORMAT OF STRAP II's OUTPUT ON DISK

STRAP II will place the first block of listing and the first block of binary output in the arcs specified. The first word of each arc written will be an index word that has the following configuration:

Format of Header Index Word Before Each Arc of STRAP Output

| | |
|---|---|
| Bits 0 - 17 | contains the relative arc number where the current output is continued. |
| Bits 18 - 19 | not used. |
| Bit 20 | if on, indicates present record is in binary code. |
| Bit 21 | if on, indicates present record is in A6 code. |
| Bit 22 | if on, indicates present record is in A8 code. |
| Bit 23 | not used. |
| Bit 24 | if off, indicates present record is listing; if on, binary output. |
| Bit 25 | if on, indicates this is the last record of the type of output specified by bit 24. |
| Bit 26 | if on, indicates present record is in IQS code. |
| Bit 27 | not used. |
| Bits 28 - 45 | contains the number of 'unit records' per arc (lines per arc, cards per arc). |
| Bits 46 - 63 | in the case of a listing arc, contains the relative arc number of the previous arc of listing. |

The second word of each record will be a "dummy" word. The data begins in the third word.

STRAP will leave the listing and binary data "sandwiched" or interspersed, on the disk in records of arc lengths.

When assembly is complete, the data in word 9 of the Communication Record will be placed in word 8, with bits 00 - 18 containing the relative arc number of the last arc of listing.

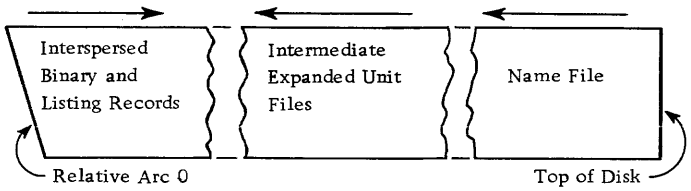



Figure 2. STRAP II's Output on Disk

## CONVENTIONS ADOPTED IN THE DESCRIPTION OF THE PHASES AND OF THE SUBROUTINES

Underlining (in the context only and excluding from consideration that in titles) will be used exclusively to refer to locations in the STRAP II program. An underlined word, all in caps, refers to a logic area whose first symbolic location generally is the name given to the particular subroutine. An underlined word with only the initial letter capitalized refers to an indicator, buffer, index word, etc. .

The term unit, unless otherwise specifically qualified, will be used to refer to the intermediate expanded instruction unit record built up by STRAP II for each input instruction.

The purpose of particular logic areas (loosely called subroutines during the descriptions) will be included with each index of subroutines rather than with the write-ups of the subroutines.

Specific mention of which error situations are detected in the subroutines is not generally made in the subroutine write-ups. Instead the subroutine that usually detects the conditions causing a particular error message is included with the itemization of the error messages in the section on serviceability aids.

A sub-procedure or logic area which performs a specific function is broadly referred to as a subroutine.

The main flow of Pass 1 is referred to as MAIN; the main flow of Pass 2 is referred to as Pass 2.

Even though most of the sub-procedures are just sections of either main flow process, reference to the performance of a particular function will generally be made by using the name of the sub-procedure, e.g., referring to the SYN procedure in main flow of Pass 1 by MQS, instead of by MAIN.

The variable length entry (vle) portion of the symbol table will be referred to as the vle table; an entry in this table will be referred to as a vle.

When reference in the text is made to an input character, the character will be represented in the description by the character's standard graphic (or coding symbol) enclosed in a circle, e.g., the ; .

Diagrams and formats for the most part are included in the Appendix rather than in the text itself.

## INTRODUCTION TO THE PHASE ACTION IN STRAP II

STRAP II consists of two passes, the second of which has two parts. (The value of each symbol is not available at the end of Pass 1 because a program to be assembled by STRAP II is allowed to have in the data description and/or address field(s) symbols which have not been previously defined in the program. Thus the establishing of values for the binary output and the settings of the location counter must be done during a two-part Pass 2.)

### Pass 1

Pass 1 processes the input statements and builds up an intermediate expanded instruction unit for each statement in a buffer which is written out, when filled, onto the disk. Each expanded instruction unit is a record at least eight words in length. The first of these eight words is a control word for locating and chaining the units. (A unit occupies the same area each time it is in memory.) The next seven words is the fixed portion of the unit, fixed in the sense that these seven words have a format in which information both about the instruction and for the listing is saved in particular fields of the unit. The format of the fixed portion of the unit is defined in terms of its relative locations by the symbols beginning with Zi. The absolute address of the current expanded instruction unit is in the value field of $6 during Pass 1 and in the value field of $3 during Pass 2. Following the initial eight words will be the coded expressions (or special configurations as that made up for the PUNSYM symbols) associated with the instruction. If the data description was unable to be evaluated, its coded expression will be first. Then will follow the coded expression(s) for the address field(s) of the instruction. The print image of the statement field to appear on the listing occupies the final portion of the unit. Access to information in a particular unit is possible only when that unit is being processed because in general the units are not all in memory at the same time. (The name which appears on the listing with the statement print image contained in the unit is saved in another file.)

Information is put in the unit as each field in the input instruction is analyzed by main flow. Thus the coded expressions will be stacked in the same order as that of the fields they represent. Since the total length of the coded expressions is unknown during the time the characters of the statement field are being analyzed, the characters of the variable length statement field are saved in another buffer and collectively inserted into the unit after all the fields in the input instruction have been analyzed.

If a name is associated with the instruction, Pass 1 makes a symbol table entry in the STRAP II symbol

table which remains in memory during the entire assembly. The symbol table entry recording the occurrence of a name with the first instruction of a card block is made of two parts: the fixed or dictionary portion contains at least part of the symbol and the address of its variable length entry, the variable length entry portion contains the remaining characters, if any, of the name, information about the symbol such as its data description, and its value or the coded expression of the value of the symbol. The format of the function section of the variable length entry portion of a symbol table entry is defined in terms of its relative location by the symbols beginning with Zst.

Another dictionary table, the one for the tail, also uses the same vle table for the remainder, if any, of its dictionary entries. Furthermore one type of entry - namely the one built up for any possible candidate for the highest or lowest assembly value - is recorded only in the vle table with no portion in a dictionary table. Other entries in the vle table which do not have a corresponding dictionary portion include old vle portion(s) of any multiply-defined symbol with no contradictions and any dimension entry made in addition to its corresponding regular symbol table entry for a name on a DR, DRZ, or SYN. (Material marked with an ID character and stacked chronologically in the unordered vle table is fetched by table management with a linear search. Material alphabetically inserted in any ordered dictionary table is fetched with a binary search.) Note in regard to the symbol table, each entry in the dictionary portion table must have a corresponding vle portion because of the way a unit marked as having a name is coordinated with its symbol table entry for Pass 2 processing.

Note the information for a unit is built-up during the analyzation of its instruction, while the symbolic argument for the dictionary portion of the regular symbol table entry is collected over the analyzation of all the instructions in the card block. At the end of the card block the symbol information is coordinated, and the symbol table entry is made. The information common to the unit and its corresponding symbol table entry, if any, are the name check digit and the contents from the absolute and symbolic counters, i.e., the counters that determine the value of the location counter.

After Pass 1 there is the first iteration of the symbol table, which both resolves the data description associated with each symbol and tries to establish values for the symbols, particularly the values of the SYN chains.

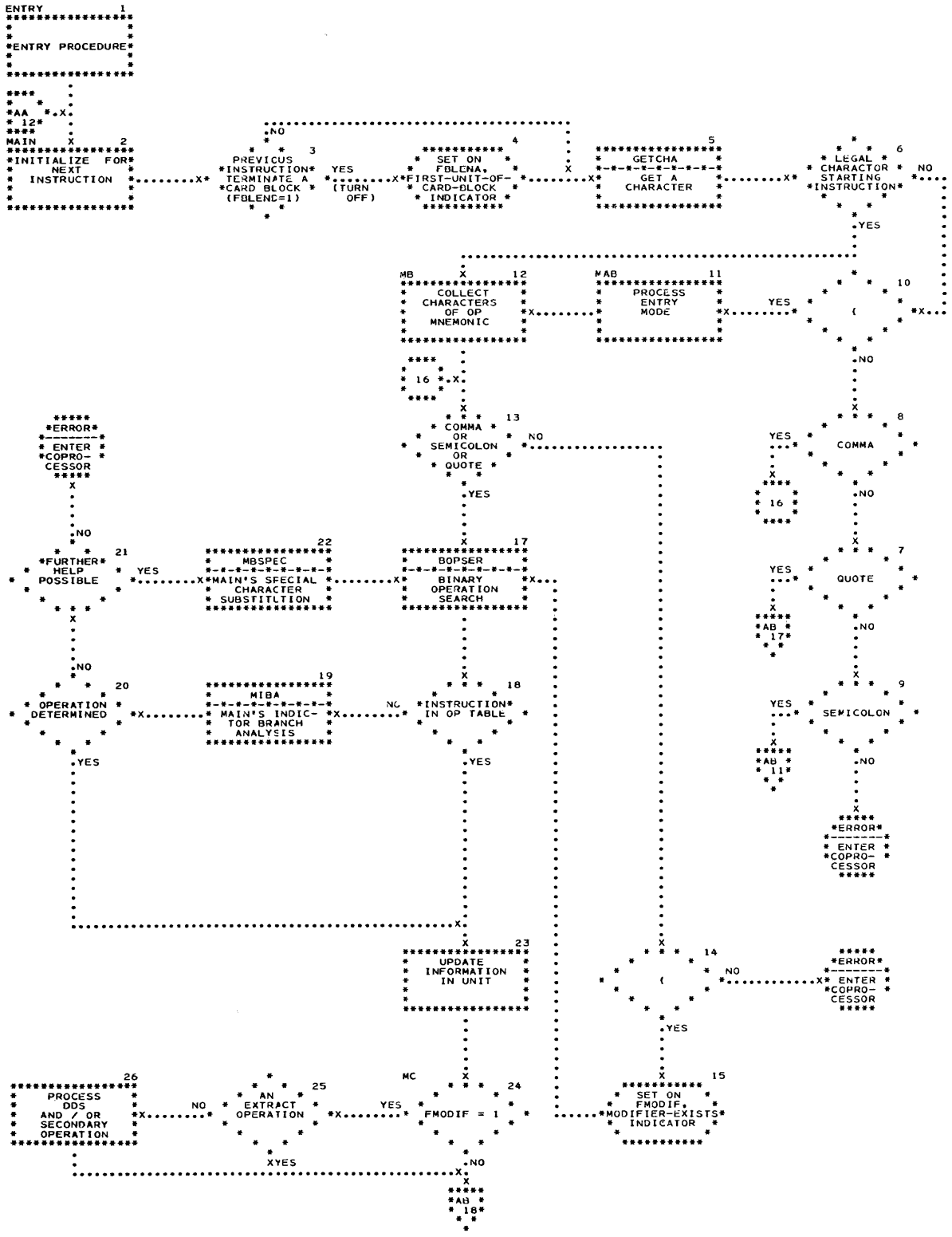
Pass 2

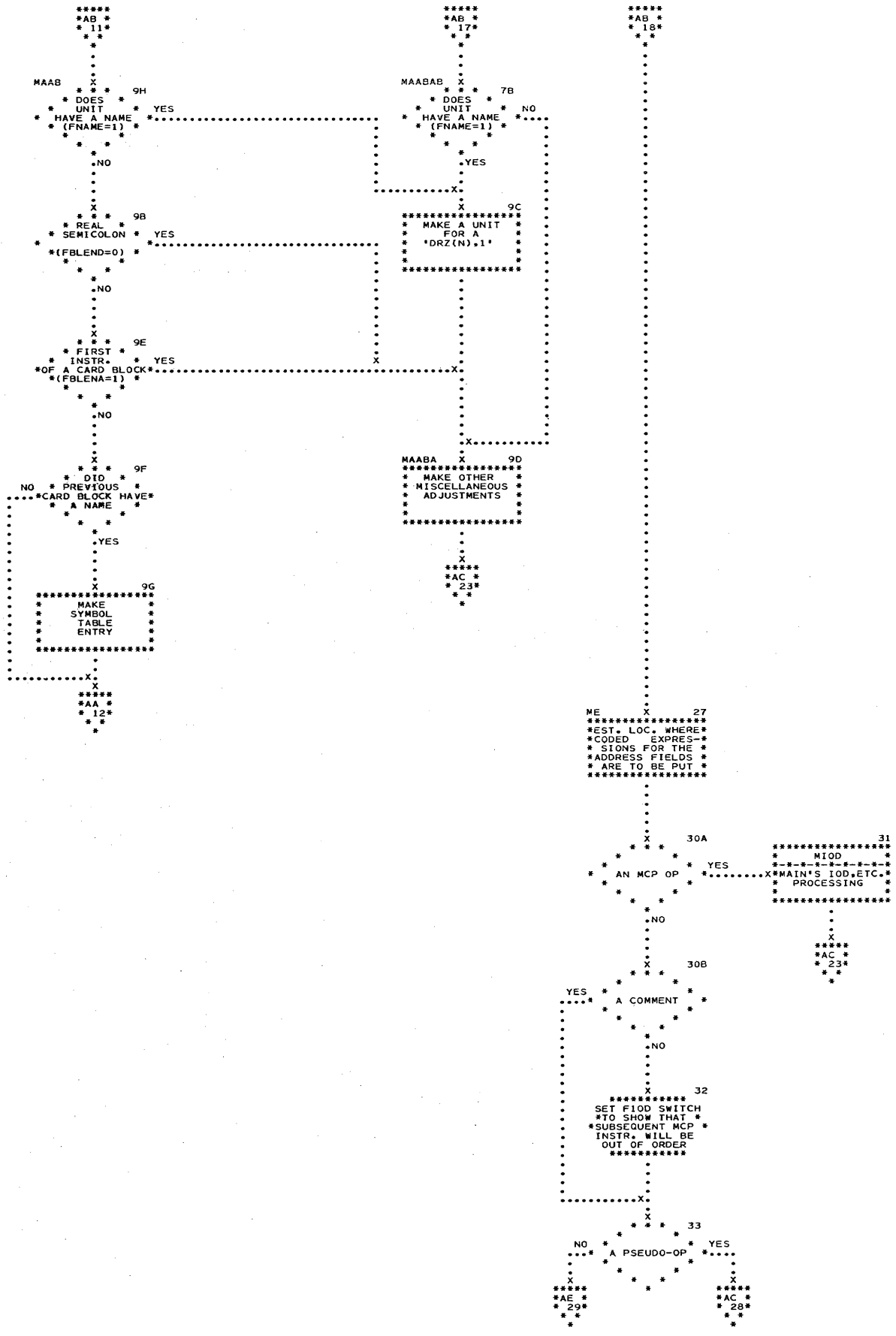
Pass 2a assigns location counter values to the instructions and attempts to assemble them. Following Pass 2a there is a second pass of the symbol table which establishes the absolute value for each symbol in the symbol table. As in Pass 1, each of the units passes through Pass 2a singularly and then through Pass 2b. When each unit reaches the end of Pass 2b, Pass 2b gives it to the output program for its inclusion in the final documents.

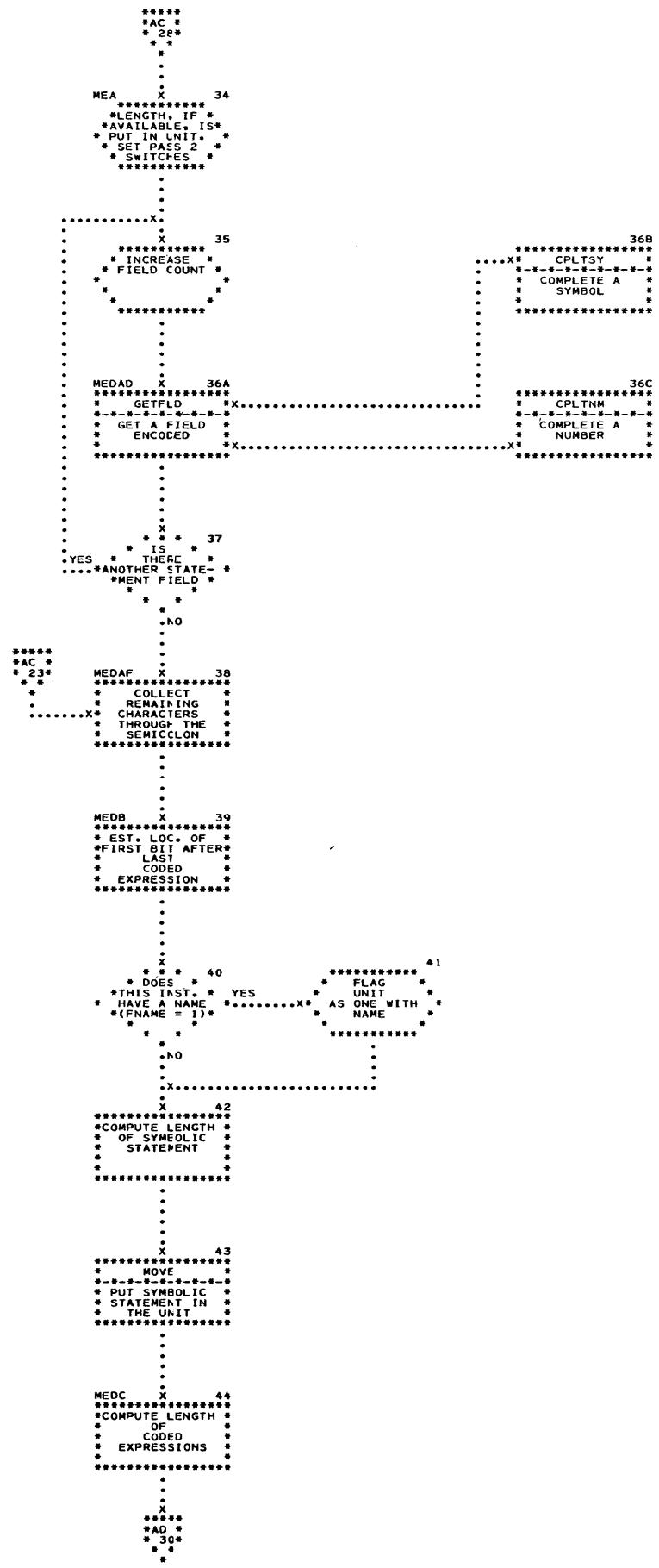
To establish each location counter value, STRAP II maintains an absolute and symbolic value, each in its own index register during Pass 2. The contents of these counters are put into the unit and into the function part of the variable length entry portion of the corresponding symbol table entry, if any, during the final processing in Pass 2b just before the output procedure prints and punches the documents for the instruction represented in the unit. The field in the unit reserved for the contents of the absolute counter is the source of the location counter value for the output procedure in Pass 2b.

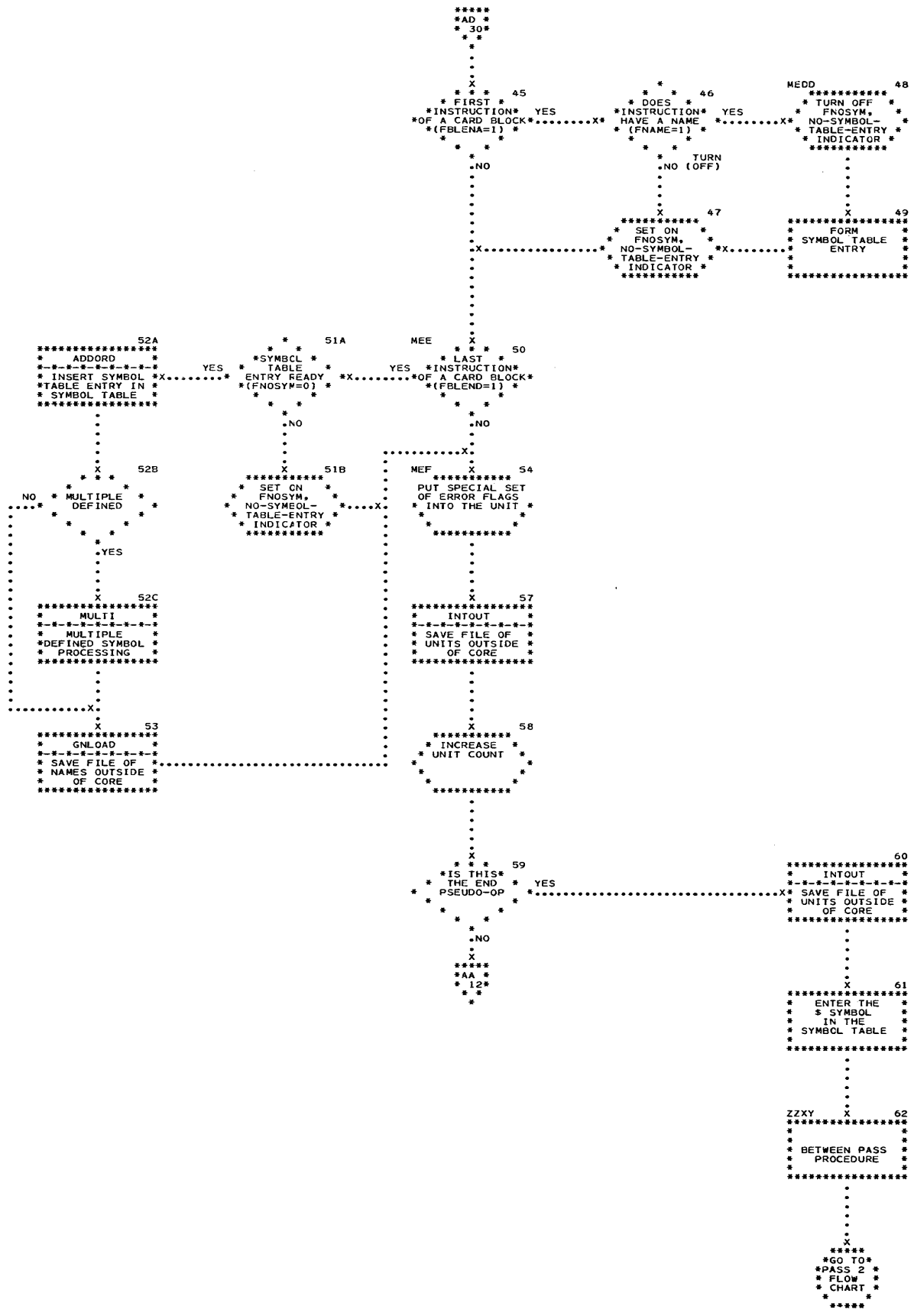
Two counters have to be maintained during Pass 2a because there are occasions during Pass 2a when the value of the yet unevaluated programmer's symbol is needed in establishing a particular location counter value. When this situation arises, the absolute counter is set to zero, a unique STRAP II special symbol is put in the symbolic counter, and an entry made in the symbol table for the special symbol. This STRAP II symbol is a three A8 character, the first character of which is the all ones A8 character reserved for STRAP II. (Taking advantage of the fact that each of these relative address symbols will necessarily be the currently last symbol in the symbol table, Pass 2 has the symbol inserted unordered into the symbol table since fewer subroutines in table management are involved in the unordered type of insertion.)

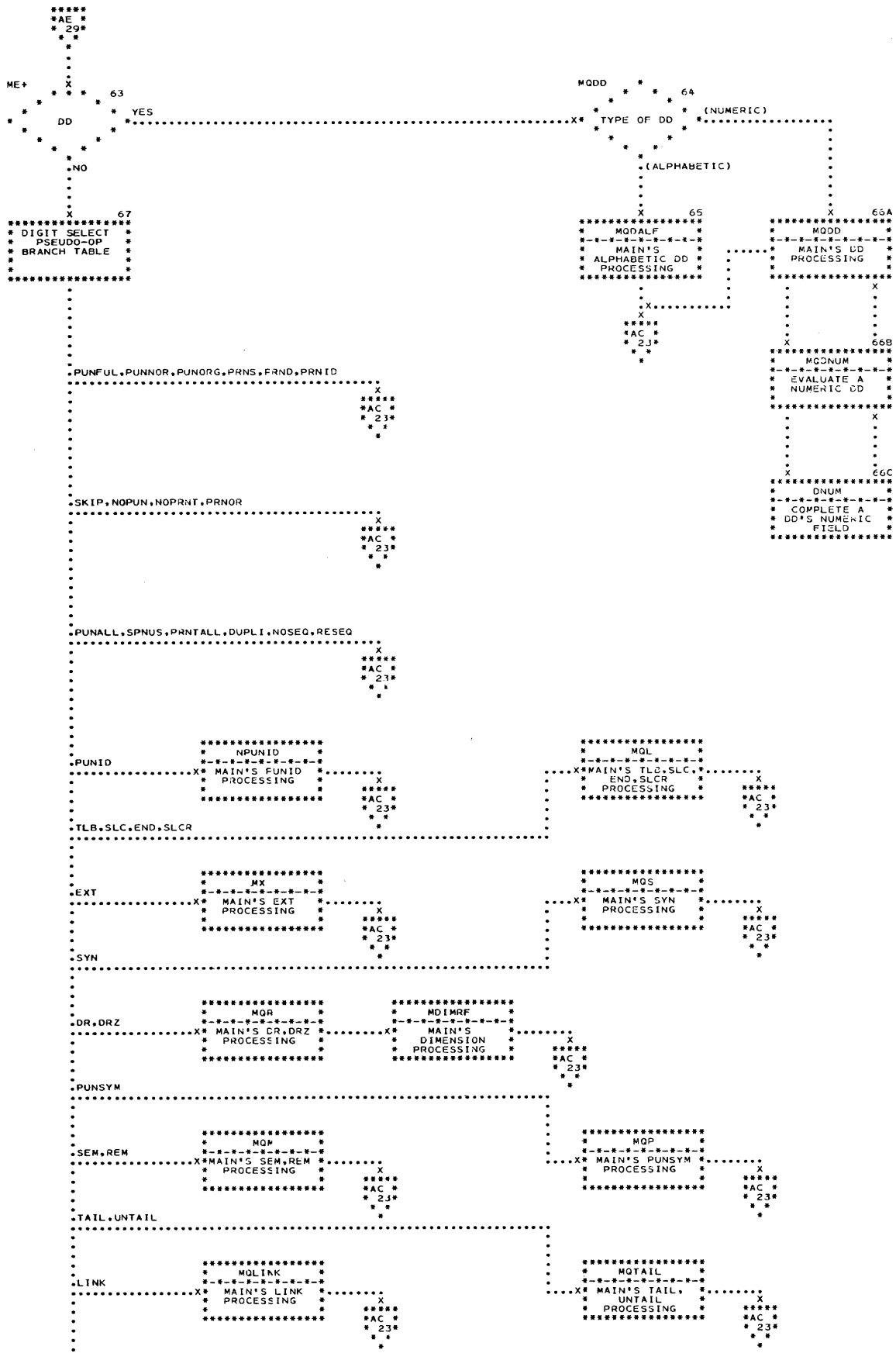
In function part of the variable length entry portion of the STRAP II special symbol, a SYN-like coded expression is built up to represent what the location counter value should be. This STRAP II special symbol will be evaluated during the second iteration. Once a STRAP II symbol is put in the symbolic counter, the STRAP II symbol remains there while the absolute incrementing is done on the absolute counter. If Pass 2a reaches another instance where the absolute increment is not available for updating, the absolute counter is set to zero again, a new STRAP II special symbol is put in the symbolic counter, and a coded expression built up in the vle


ENTRY 1
*ENTRY PROCEDURE*
*AA 12*
MAIN 2
*INITIALIZE FOR NEXT INSTRUCTION*
3
PREVIOUS *INSTRUCTION* TERMINATE A *CARD BLOCK * (FBLENC=1)
YES (TURN OFF)
4
SET ON FBLENA, FIRST-UNIT-OF-CARD-BLOCK INDICATOR
5
GETCHA
GET A CHARACTER
6
LEGAL CHARACTOR STARTING *INSTRUCTION*
NO
YES
MB 12
COLLECT CHARACTERS OF OP MNEMONIC
MAB 11
PROCESS ENTRY MODE
YES
10
16
13
COMMA OR SEMICOLON OR QUOTE
NO
YES
*ERROR* ENTER *COPROCESSOR
8
COMMA
7
QUOTE
9
SEMICOLON
*AB 17*
*AB 11*
*AB 16*
21
*FURTHER* HELP POSSIBLE
YES
22
MBSPEC
*MAIN'S SPECIAL CHARACTER SUBSTITUTION
17
BOPSER
BINARY OPERATION SEARCH
20
OPERATION DETERMINED
19
MIBA
MAIN'S INDICATOR BRANCH ANALYSIS
NO
18
*INSTRUCTION* IN OP TABLE
23
UPDATE INFORMATION IN UNIT
14
NO
26
PROCESS DDS AND / OR SECONDARY OPERATION
NO
25
AN EXTRACT OPERATION
YES
MC
24
FMODIF = 1
15
SET ON FMODIF, *MODIFIER-EXISTS* INDICATOR
XYES
*AB 18*


*AB * 11*
MAAB
9H
DOES UNIT HAVE A NAME (FNAME=1)
YES
NO
9B
REAL SEMICOLON (FBLEND=0)
YES
NO
9E
FIRST INSTR. OF A CARD BLOCK (FBLENA=1)
YES
NO
9F
DID PREVIOUS CARD BLOCK HAVE A NAME
NO
YES
9G
MAKE SYMBOL TABLE ENTRY
*AA * 12*
*AB * 17*
MAABAB
7B
DOES UNIT HAVE A NAME (FNAME=1)
NO
YES
9C
MAKE A UNIT FOR A 'DRZ(N),1'
MAABA
9D
MAKE OTHER MISCELLANEOUS ADJUSTMENTS
*AC * 23*
*AB * 18*
ME
27
*EST. LOC. WHERE* *CODED EXPRES-* * SIONS FOR THE * *ADDRESS FIELDS * * ARE TO BE PUT *
30A
AN MCP OP
YES
NO
31
MIOD
*MAIN'S IOD,ETC.* PROCESSING
*AC * 23*
30B
A COMMENT
YES
NO
32
SET F1OD SWITCH *TO SHOW THAT * *SUBSEQUENT MCP * INSTR. WILL BE OUT OF ORDER
33
A PSEUDO-OP
NO
YES
*AE * 29*
*AC * 28*


AC 28
MEA
34
LENGTH, IF AVAILABLE, IS PUT IN UNIT. SET PASS 2 SWITCHES
35
INCREASE FIELD COUNT
36B
CPLTSY
COMPLETE A SYMBOL
MEDAD
36A
GETFLD
GET A FIELD ENCODED
36C
CPLTNM
COMPLETE A NUMBER
37
IS THERE ANOTHER STATE- MENT FIELD
YES
NO
AC 23
MEDAF
38
COLLECT REMAINING CHARACTERS THROUGH THE SEMICLON
MEDB
39
EST. LOC. OF FIRST BIT AFTER LAST CODED EXPRESSION
40
DOES THIS INST. HAVE A NAME (FNAME = 1)
YES
41
FLAG UNIT AS ONE WITH NAME
NO
42
COMPUTE LENGTH OF SYMBOLIC STATEMENT
43
MOVE
PUT SYMBOLIC STATEMENT IN THE UNIT
MEDC
44
COMPUTE LENGTH OF CODED EXPRESSIONS
AD 30


AD 30
45
FIRST INSTRUCTION OF A CARD BLOCK (FBLENA=1)
YES
NO
46
DOES INSTRUCTION HAVE A NAME (FNAME=1)
YES
NO
TURN (OFF)
47
SET ON FNOSYM, NO-SYMBOL-TABLE-ENTRY INDICATOR
MEDD
48
TURN OFF FNOSYM, NO-SYMBOL-TABLE-ENTRY INDICATOR
49
FORM SYMBOL TABLE ENTRY
MEE
50
LAST INSTRUCTION OF A CARD BLOCK (FBLEND=1)
YES
NO
MEF
54
PUT SPECIAL SET OF ERROR FLAGS INTO THE UNIT
57
INTOUT SAVE FILE OF UNITS OUTSIDE OF CORE
58
INCREASE UNIT COUNT
59
IS THIS THE END PSEUDO-OP
YES
NO
AA 12
52A
ADDORD INSERT SYMBOL TABLE ENTRY IN SYMBOL TABLE
51A
SYMBCL TABLE ENTRY READY (FNOSYM=0)
YES
NO
51B
SET ON FNOSYM, NO-SYMBOL-TABLE-ENTRY INDICATOR
52B
MULTIPLE DEFINED
NO
YES
52C
MULTI MULTIPLE DEFINED SYMBOL PROCESSING
53
GNLOAD SAVE FILE OF NAMES OUTSIDE OF CORE
60
INTOUT SAVE FILE OF UNITS OUTSIDE OF CORE
61
ENTER THE $ SYMBOL IN THE SYMBOL TABLE
ZZXY
62
BETWEEN PASS PROCEDURE
GO TO PASS 2 FLOW CHART


AE 29
ME+
63
DD
YES
NO
MQDD
64
TYPE OF DD
(NUMERIC)
(ALPHABETIC)
65
MQDALF
MAIN'S ALPHABETIC DD PROCESSING
66A
MQDD
MAIN'S DD PROCESSING
66B
MQDNUM
EVALUATE A NUMERIC DD
66C
DNUM
COMPLETE A DD'S NUMERIC FIELD
67
DIGIT SELECT PSEUDO-OP BRANCH TABLE
AC 23
PUNFUL,PUNNOR,PUNORG,PRNS,PRND,PRNID
SKIP,NOPUN,NOPRNT,PRNOR
PUNALL,SPNUS,PRNTALL,DUPLI,NOSEQ,RESEQ
PUNID
NPUNID
MAIN'S PUNID PROCESSING
MQL
MAIN'S TLB,SLC,END,SLCR PROCESSING
TLB,SLC,END,SLCR
EXT
MX
MAIN'S EXT PROCESSING
MQS
MAIN'S SYN PROCESSING
SYN
DR,DRZ
MQR
MAIN'S DR,DRZ PROCESSING
MDIMRF
MAIN'S DIMENSION PROCESSING
PUNSYM
SEM,REM
MQM
MAIN'S SEM,REM PROCESSING
MQP
MAIN'S PUNSYM PROCESSING
TAIL,UNTAIL
LINK
MQLINK
MAIN'S LINK PROCESSING
MQTAIL
MAIN'S TAIL, UNTAIL PROCESSING

portion of the new symbol table entry to indicate the cascading of the relative location STRAP II special symbols. The second iteration will establish the values of these STRAP II special symbols along with the values of the other symbols in the symbol table, Pass 2b will then combine the absolute and evaluated symbolic value, if any, so that each actual location counter value can be put into the absolute field of each unit for the output program.

The attempt to assemble takes place when the decoding procedure gets values which can be selectively inserted into the skeleton word provided in the unit. This is done during Pass 2a if all datums in the coded expressions for the instruction fields can be evaluated. Otherwise, the filling in of the binary output is done during Pass 2b. If while evaluating a coded expression the evaluating subroutine cannot establish a value, Pass 2a still has the entire coded expression scanned (by a special re-entry into the evaluating subroutine) so that any undefined symbols used in the coded expression can be detected and inserted in the symbol table.

## DESCRIPTION OF THE ACTION INDICATED ON THE FLOW CHART OF PASS 1

### Box 1

The entry procedure for STRAP II includes the following actions:

1. The entry procedure saves the current setting of the time clock. In Pass 2, OUTPUT uses this value to mark the binary cards; the end-of-assembly procedure uses it to compute the total time of the assembly.
2. The entry procedure writes the current pass indication on the numeric display lights of the console.
3. The entry procedure also has MCP change two of its IOD addresses, the one for the console and the one for the disk.
4. By checking $UB, the entry procedure determines the size of the machine, and then sets the limits of the STRAP II internal symbol table accordingly.
5. The ERR and VALUE subroutines are initialized now.
6. During the assembly many of the index registers have fixed uses. The refill field of $15 has a special function and is now set up with the location of STRAP II's table of exits for maskable interrupts.
7. After having fetched the Communication Record from MCP, the entry procedure determines STRAP II's position in the chain. If STRAP II is first in the chain, the entry procedure sets the Communication Record to indicate input is System Input. In any event, the entry procedure goes to MAIN to begin the processing of the input.
8. The ZM bit and the EXE bit in the mask register are turned on so that MCP will give these interrupts to STRAP II.

### Box 2

The action of this box is performed at the beginning of the Pass 1 processing for each new instruction.

Buffers are reset: The two word buffer, Bem, containing the entry mode control data is zeroed. The collection buffer, Bop, for the primary op is blanked. Thus, Bop will contain the proper background characters for the later table-lookup. The field counter which the error message routine uses in locating the field of the error is set to zero. On the other hand, the unit counter, initially defined as one, is incremented by Pass 1 during its end processing of the former unit rather than during the initialization of the new unit about to be processed. The fixed format part of the current expanded instruction unit is set to zero. (See Appendix A.) This fixed format will contain the specific information about the instruction, the binary output, the dimension reference address, the ID, etc. .

Indicators and flags are reset: The if-modifier and if-pseudo-dds indicators are initially set off. (These 'if' indicators are not in the unit.) The if-all-evaluated indicator in the fixed format part of this unit is now initially set on. The mask containing the five error symbol indicators and the suppress-error-message indicator is zeroed. The bit setting of these six indicators is put into the fixed format part of the unit during the final processing of the unit in each phase. Also, the indicators, flags, and a radix for GETFLD are reset.

Control index words are reset: Bop's controlling index word, Bopx, is reset. The index word, Bsystx, which controls the collection buffer for the statement field characters that shall appear on the listing with the instruction is reset. (GETCHA collects the characters; MAIN puts them into the unit.) The index word, Ibuffw, locates available area in the buffer reserved for the units. Since Ibuffw is now addressing the starting location of a new unit, MAIN adjusts this address to a full word location and saves this address in $6. (The fixed format fields in the unit are referenced by $6 during Pass 1.) Next MAIN updates Ibuffw to address the available area after the fixed format where the coded expression(s), if any, followed by the statement field characters will be put. This location of the starting point of the coded expression(s) is saved for the later computation of the total length of the coded expression(s) in this unit. The upper limit of the buffer reserved for the units is put in GETFLD's index word, Bcfxtx, since this is one of the index words that have to be set up for using the encoding subroutine. Besides knowing where to put

coded expression(s), the encoding subroutine, GETFLD, must also know the upper limit of the buffer containing the encoding. Since GETFLD will probably put more than one coded expression into the unit for this instruction, the upper boundary is set this one time to include all the uses of GETFLD for this purpose. Bcfxtx has to be reset for each instruction because GETFLD can be asked to put coded expression(s) in places other than in the unit (e.g.: in variable length entry portion of the symbol table for a SYN).

Finally, the contents of the identification field (columns 73-80) is inserted in the space provided in the fixed format part of the unit for the card ID which will be later printed on the listing. Note an instruction which is specified over several cards is assigned the ID of the card on which the instruction started.

STRAP II does not indent on the listing the second and subsequent instructions contained in a single card image; if there is a different ID on each card image as is the case when a sequence number is used in columns 73-80, each of the instructions started on the same card image will have the same ID on the listing.

Boxes 3, 4

One of the responsibilities of the subroutine, GETCHA, is to inform Pass 1 when the end of a card block has been reached by putting a 'false' semicolon (false in the sense that it did not come from the card image) in the value field of $3 and by turning on the if-block-end indicator, Fblend. If Fblend is on when MAIN is starting the processing of a new instruction, then the previous instruction terminated a card block and the present instruction is beginning a new card block. (Fblend is initially on.) If Fblend is on at this point, then this instruction is marked eligible for a name by setting the Fblena indicator on. Fblend is turned off here as a result of the test. During the final processing of a unit in Pass 1, if Fblena is on, then the symbol table entry, if any, can be prepared and set up; if Fblend is on, then the name and the symbol (name without blanks for internal processing) associated with the card block can be entered into each of its respective files.

Box 5

The get-a-character subroutine, GETCHA, gives to main flow the next character from the statement field of the card image. Each new character - now in the A8 processing code of STRAP II - is put in the value field of index register three during Pass 1. (See section on serviceability aids.) GETCHA does not give blanks to main flow unless main flow has turned on the indicator, Gcaz.

Main flow interrogates characters in the following manner. For each IBM card code character and for specific grouping of characters, there is a 256-bit question block in which the bit(s) corresponding to the character(s) indicated have been permanently masked on. The symbolic location of each string of bits is given by a symbol beginning with Qc. The relative locations within the question block is addressed by the A8 character in the value field of $3.

Ex.: Is this character a "$"? If yes, branch to Mb area.

```
BB, Qcdola($3), Mb
```

Since Pass 1 analyzes most of the characters from the source input through these question character blocks, a pre-processor can send a special A8 character through STRAP II by setting the high order two bits of the A8 character to one and the corresponding question character block accordingly. A special character, thus set up, will be interpreted as an alphameric character by the question character blocks.

GETCHA also collects the characters from a name field in the name buffer and in the symbol buffer each of whose contents is picked up during the end-of-card-block procedure for the name and symbol table entries. Also, GETCHA builds up an image (for the listing of the statement field of this instruction) in a buffer which MAIN later transfers to the unit. The statement field of this instruction cannot be directly built up in the unit because both the coded expressions, if any, which precede the symbolic statement in the unit and the symbolic statement are not a constant length.

Box 6

The only characters that are allowed to start an instruction are: all legal op characters, (, ;, ', $, and a ,. (This test of the first non-blank character of an instruction is a good example of STRAP II's use of the question character blocks.)

Boxes 7, 7B, 9C, 9D, 38

If MAIN receives a comment character as the first character of the first instruction of a named card block, then MAIN makes up a "DRZ(N), 1" instruction for the current unit. Whether the unit has a name or not, for a comment character Pass 1 continues to branch to GETCHA until GETCHA gives the terminating semicolon character to main flow. The only processing necessary for the intermediate characters of the comment is that they be saved for the final listing

in Pass 2b. GETCHA saves the characters associated with each instruction in a symbolic statement buffer which Pass 1 transfers into the unit during the end-of-instruction processing in MAIN.

Box 8

For a comma character (a null op) Pass 1 transfers to that part of the main flow where up to eight characters of the op have already been collected in Bop. Note: since the buffer was blanked out in the initialization, Bopx is already set up for the look-up of the null op in the primary op table.

Boxes 9-9H

For a semicolon character in a named card block, MAIN moves a precoded - DRZ(N), 1 - unit into the current unit. For a semicolon character in an unnamed card block, a comment unit is set up. This ultimately will produce a blank line on the listing. However the setting of the ID field in the unit prior to this move is saved and restored. Because the precoded format has all the information needed for the instruction, practically all the intermediary action between this box and box 38 is not needed. However, three things must be done before the transfer to box 38: First, $5 is set up with the location of the appropriate op index. Second, since the usual encoding of GETFLD will not take place for this DR, index word Bcfldx must be adjusted so that later redefinition of Ibuffw by Bcfldx will be correct. Third, Fiod is turned on to indicate a non-MCP instruction has occurred.

In summary, if the first character of a new instruction is a semicolon, one of the following three situations is possible:

| Setting of Fblend | Setting of Fblena | Comment |
|---|---|---|
| off | - | More than one semicolon has been specified after an instruction which is not the last instruction in this card block. STRAP II makes up a -- DRZ(N), 1 -- instruction for this unit. |
| on | on | A semicolon has been specified as the first character of this card block STRAP II makes up either a -- DRZ(N), 1 -- instruction in the unit if it has a name or a comment if unnamed. |
| on | off | This semicolon is not from the symbolic input but is the "false" semicolon from GETCHA. A semicolon (already processed) has been specified unnecessarily after the last instruction in the previous card block. If this immediately previous card block has a name, MAIN makes the entry into the symbol table now because MAIN knows that the card block has finally been completed. |

Boxes 10, 11

For a (, Pass 1 determines the type of entry mode before having the characters of the op collected. A two word buffer at Bem, which holds the information about any entry mode and radices on a DD, is built up. The second index word contains the parameter, if any, specified before the op; the first index word, the radix to be used on the current D field under process. In each index word the value field is reserved for the terminating character of an alphabetic DD; the twenty four bits, 36-60, for the radix; bits 61-63, Bemt, Bemu, Bemv, for the type of DD.

| Setting of Bemt | Setting of Bemu | Setting of Bemv | Comment |
|---|---|---|---|
| 1 | 0/1 | 0/1 | alphabetic entry mode |
| 1 | 0 | 0 | A entry mode |
| 1 | 0 | 1 | CC entry mode |
| 1 | 1 | 0 | IQS entry mode |
| 0 | 0 | 0 | no entry mode given |
| 0 | 0 | 1 | Fn entry mode |
| 0 | 1 | 0 | (r) radix |
| 0 | 1 | 1 | (.n) parenthetical integer entry mode |

This information is picked up later when MQDD or MQDALF processes the DD.

Boxes 12-15

Using GETCHA, MAIN collects the characters of the op mnemonic, each being tested for legality. Up to

eight characters are collected or up to the next separator character. If a ( follows the op, the if-modifier-exists indicator, Fmodif, is set on, so the dds or $op_2$ processing will be done after the table lookup of the primary op. Modif being on indicates to MAIN that there are data description, secondary op, or EXTract parameters yet to be considered before the address field of the instruction can be encoded.

Boxes 17-22

The binary operation search, BOPSER, does the look-up on the ordered table of primary operations. (This subroutine is also used to search the table of secondary ops and the table of system symbols.) If the op mnemonic is not found the op may be one of the 'branch on indicator' type instructions which are not in the op table because of the impracticality of including all their variations. A special purpose routine, MIBA, supplies for a 'branch on indicator' type of instruction that information which would have been in an op table entry if the operation had been in the op table. If the op is not a 'branch on indicator' instruction, by using MBSPEC, subroutine MIBA tries substituting alternate characters for any which might be making the op illegal. If subsequent retries on looking up each new variation of the op fails, there is an error condition, and the unit is set up as a -- SIC, $15; BE, 0 -- instruction.

Box 23

The entry in either op table is generally the source of both the skeleton bit pattern for the instruction and the location of the "op index" (op question bit string) for the operation. However, the pseudo-op's entry in the primary op table is the source of both the "digit select" numbers assigned to the pseudo-op and the location of the op index. (This number will be used in three different phases of the assembly in determining the treatment of an individual pseudo-op.) The ambiguous op's entry in the primary op table has the skeleton bit pattern of the normalized version of the instruction, the last half of the bit pattern of the variable field length version, and the location of the op index for an ambiguous op. There is one entry (not two entries) for an ambiguous op in the table of primary operations. (STRAP II processes the unit of an ambiguous op as that of a floating point instruction until diagnosis of the data description or address fields in the instruction proves otherwise. Note in Pass 2a if the data description to be associated with an ambiguous unit is still undetermined after DECODE has gone to VALUE for the dds, DECODE assigns a VFL dds to the unit. Of course, an ambiguous op's field types are not referred to until the ambiguity is settled.)

Information from the op entry is put in the unit and the location of the op index is also put in $5 for common referencing during Pass 1.

During the processing of a unit in Pass 1 and Pass 2, $5 has the constant function of addressing the op index. For each operation there is provided a string of bits which represent yes/no answers to the questions about the op such as: Is this a variable field length op? The format of the operation question bits is defined in terms of its relative locations by the symbols beginning with Zo. The absolute location of each string of op question bits is in the first 18 bits of the function part of the op entry in the table of ops. The first 24 bits in the actual string contains the address of the first control word in the chain of control words which are used in Pass 2 to insert the address fields into the binary output saved in the fixed format part of this op's unit. The op index for a binary producing instruction which can have a secondary op also contains the code number for the secondary op. (See Appendix A.)

Ex.: Is this an immediate variable field length op? If no, branch to Mdg1 area.
BZB, Zoimm($5), Mdg1

If an operation is added to the op table, besides possible processing alterations, changes will have to be made to some or to all of the following tables: table of operations, table of op question bit strings, list of op index words, INSERT's index words, control block for BOPSER, three pseudo-op branch tables.

Boxes 24, 25, 26

Since the EXT parameters are handled in the EXT pseudo-op subroutine later, this type of case is postponed for the later treatment. However if the character after the ( is one of the characters in the question character block which contains the legal first characters for a secondary op, then MAIN collects the remaining characters of the secondary op and looks up the op in the secondary op table. Once again MBSPEC may be called upon to help. Note the format of an entry in the secondary op table differs from that in the primary op table. The secondary op may not be greater than seven characters in length since the dictionary part of a secondary op table entry contains a special leading 8-bit byte code number to identify the type of instruction which may have this secondary op. This code number is obtained from the current op index and is the first character put in the secondary op collection buffer, Bop2x. When a secondary op follows a primary op which permits a secondary op, MAIN makes up an appropriate argument for the table look-up. Thus the cases, such as -- + (BU, 3) (SEOP) and CW (V + I) -- are illegal, although in each instance the primary op does allow a secondary

op. For these operations, MAIN would attempt to look up the corresponding argument lSEOPbbb or 4V+Ibbbb, respectively, in the secondary op table. Since neither argument is in the table, the illegal op would be detected without further cross checking. (See Appendix A.)

If the op is an ambiguous-type op, the presence of a secondary op establishes the op as a variable field type op; the variable field length skeleton bit pattern is set up in the unit now; the location of the op index for a variable field length op is put in the unit and into $5.

In the processing of the data description in Pass 1, MAIN treats a specified P-mode, absolute, symbolic or absolute-symbolic data description in the following manner. MAIN has the complete symbol routine, CPLTSY, collect the characters of the P-mode symbol and put them in the area of the unit reserved for the coded expressions. This symbol becomes the coded expression for the data description of this instruction. MAIN computes the length of the P-mode symbol by subtracting the value field of index word, Ibuffw, from the value field of CPLTSY's updated index word, Bsymbx, which was set equal to Ibuffw prior to the entrance into CPLTSY. This length is saved in the fixed format part of the unit so DECODE will know how long this special type of coded expression is during the Pass 2 evaluation.

If the data description is not P-mode, MAIN has GETFLD encode the byte size and field length and put the coded expressions in the unit MAIN calls on VALUE to evaluate the coded expressions.

For the coded expression of each dds field that VALUE can evaluate now, the following occurs: Erroneously specified parameters are adjusted. The absolute value of the dds field is put in its specified field in the fixed format part of the unit. The dds-field-filled indicator is turned on. The coded expression is erased (by resetting Ibuffw) if there was no general parenthetical entry with the dds field. However, if this is a DD op and if there is a general parenthetical entry with the dds field, the coded expression of the dds remains in the unit, and Ibuffw is updated to space over this dds coded expression. (The coded expression of a general parenthetical entry will be evaluated by OUTPUT in Pass 2b.)

For the coded expression of each dds field that VALUE cannot evaluate, that is, for a symbolic byte size or for a symbolic field length, the following occurs: The appropriate symbolic-field-length or symbolic-byte-size indicator in the fixed format part of the unit is turned on. The coded expression is left in the unit for DECODE to have evaluated in Pass 2. Ibuffw is updated to space over the dds coded expression. The Zifall indicator is turned off to indicate to Pass 2a that it will have to evaluate a coded expression in this unit.

In addition, the appropriate mode indicator in the unit format is turned on, a null field length or a null byte size causes the specified number to be substituted by STRAP II.

Once again the ambiguity of an "ambiguous" op may be resolved. If at this point an explicit dds has just been detected, the appropriate changes are made to the unit, etc. .

Box 27

The index word, Bcfldx, which indicates to GETFLD where a coded expression is to be placed is now reset by Ibuffw. GETFLD will shortly start to encode the address fields which will probably go into the unit. If the coded expression for the address fields are not to go into unit, Bcfldx will be changed later appropriately.

Boxes 30A, 30B, 31, 32

Tests are made for a master control program op or a pseudo-op now because the processing of the address fields of each is handled by specialized subroutines. If it is not a master control op and not a comment, then the Fiod switch is set so any subsequent master control op will receive an error message because any later MCP instruction will be out of order. (MCP instructions that are out of order will be printed on the listing, but not punched.) Therefore, comments excepted, the initial instructions of the program being assembled by STRAP II must be master control ops if all the master control ops specified in the program being assembled are to be punched. MIOD handles the MCP ops, REEL and IOD, during Pass 1.

Box 33

See description under Box 67.

Box 34

The length of the binary output of a non-ambiguous binary producing instruction (DD and DR are handled separately) is put in the unit. Also the indicators frequently referenced in Pass 2 can be turned on now, Zifbo on because the unit has binary output; and Zifp2 on because the unit will contain coded expressions which must be decoded in Pass 2. Note both these indicators are in the format part of the unit.

Boxes 35-37

The get-a-field subroutine, GETFLD, is now called upon to encode in the unit each of the allowed address fields in this op. GETFLD performs a legality test

to detect syntatical errors in a field. If the datums in a field are combined in an illegal manner, GETFLD encodes a null coded expression which later will be evaluated as zero for the field. Since GETFLD encodes only one field per request, MAIN sets up a counter of the maximum number of times it should go to GETFLD. The initial contents of this counter is determined by the first of this op's fill index words which is located by the value field of the op index word. For each address field encountered, MAIN increments the field counter which is used by the error message subroutines to identify the field of an occuring error. Also a field counter in the format part of the unit is updated for each field encoded so DECODE will know how many times to go to VALUE to get a coded expression encoded during Pass 2. On each re-entry into GETFLD through this loop, MAIN does not have to reset Bcfxtx because GETFLD updates this index word. MAIN turns the Zifall indicator off so Pass 2a will know that there are coded expressions in this unit to be evaluated.

If GETFLD is called upon to encode more than one statement field for an "ambiguous op" then the "ambiguous op" must be of the vfl type and MAIN changes the unit, etc., accordingly.

Whenever GETFLD encounters a dollar sign character in an address field followed by a semicolon, comma, plus, minus, asterisk, or slash character, GETFLD forms a coded expression which contains at least the two datums to represent the dollar sign character. One datum is a symbolic datum containing the $ symbol itself represented internally by the STRAP II special symbol $00. The 24 bits occupied by the $00 configuration now, will later contain the contents of the symbolic portion, if any, of the associated location counter value. The second datum is an absolute datum presently containing 24 bits of all zeros but eventually holding the absolute portion of the location counter value. The operator to be later used with these two datums during the decoding is set up as a "L and plus" operation, so that the two datums will be combined to give the bit value of the $ setting, i.e., the present setting of the location counter value when the value of the field is established by VALUE.

For such cases, MAIN enters the $ symbol into the symbol table after all the units have been processed in Pass 1. The symbol entry will be for the symbol $00, with the function portion of the variable length entry part of the entry containing a SYN-like coded expression made up of two datums, one for the symbolic portion and one for the absolute portion of the current Pass 2 setting of the running location counter value (maintained in $6 and $7 respectively during Pass 2).

During Pass 2 in the initial processing of each unit, Pass 2 OR's into the fields (reserved for the symbolic and absolute portions of the location counter in the function portion of the $00's variable length entry) the current setting of $6 and $7 respectively.

Thus, later whenever VALUE is called upon by DECODE to establish the value of a coded expression in the unit and VALUE encounters the symbolic datum $00, VALUE recognizes the $ situation and does the following:

1. VALUE looks up the $00 in the symbol table to obtain the current absolute and symbolic values to be associated with this $.
2. VALUE inserts these values into the corresponding absoute and symbolic datums of the coded expression in this unit.
3. If there is no symbolic portion in the current contents of the $00 symbol table entry, VALUE changes the symbolic field in the unit's coded expression from $00 to all zeros. Later if VALUE has to look at this coded expression again, VALUE makes a special check for a symbolic datum of all zeroes, so that VALUE will ignore this type of symbolic datum rather than looking it up in the symbol table.

Box 38

After MAIN has processed all the characters contributing to the instruction itself, the comment character (or any characters of the instruction that are to be treated as a comment) must now be processed. Their only processing consists of having them included in the symbolic statement buffer being built up by GETCHA. MAIN accomplishes this by relooping into GETCHA until GETCHA gives MAIN a semicolon in the value field of $3. This semicolon may be either one picked up from the card image or the "false" semicolon indicating an end of card block. Later the symbolic statement buffer, i.e., the picture of the statement field of the card image, is transmitted into the unit.

Box 39

Using GETFLD's index word, Bcfldx, MAIN updates Ibuffw to address the first bit after the coded expressions with which GETFLD has been loading the unit under the control of the index word Bcfldx. Note GETFLD does not update Ibuffw; the user program must do this. Just as Ibuffw was used to locate the initially available area in the unit for the coded

expression, so Ibuffw must be updated to locate available area in the unit for the symbolic statement.

Boxes 40, 41

At this point if this is the first unit of a card block and has a name, the if-a-name indicator, Zifname is turned on and the name check digit is stored in the unit. The name check digit is one of the built-in checking features in STRAP. The check digit is carried in the symbol table entry and also in the name field entry. The synchronous matching of these files in Pass 2 is cross-checked by the use of the check digit. Note the DDI and SYN instructions, when used, must each have a name.

Boxes 42, 43

The length (count of 8-bit characters) of the symbolic statement which GETCHA has built up with each of the statement field characters in this operation is now computed and saved in the format part of the unit so OUTPUT in Pass 2b will know the length of the statement field to be printed on the listing.

The index word, Ibuffw, is rounded to a full word location so that the symbolic statement will go into the unit at a full word location, except for a comment card block and a B-type card block, the symbolic statement will not necessarily be a true image of the statement field of the op. The blanks following the last non-blank character in the statement field do not go into the statement buffer. Furthermore, MAIN adjusts the symbolic statement to eliminate the semicolon from the print image stored in the unit. After the symbolic statement has been moved into the unit, Ibuffw is updated to address available area for the next unit since no more data will go into the current unit.

Box 44

The bit length of the encoded fields in the unit is also computed and saved in the unit so OUTPUT in Pass 2 will be able to locate the statement field characters in the unit. (In Pass 2, Location of unit in the value field of $3 + size of fixed format + bit length of encoded fields = location of statement field characters.)

Boxes 45-53

Now if the unit is the first unit of the card block and if it has a name, a symbol table entry must be set up. Furthermore, if this is the end of a named card block (only at the end of a card block can an MAIN be sure that the entire name has been collected), MAIN makes an entry into the symbol table which remains in memory throughout the assembly. If the symbol collected from the name field of the card block is in the sphere of influence of a previous TAIL pseudo op still in effect, a NOP/Branch instruction before the calling sequence to ADDORD will be a Branch as set previously by MQTAIL. The internal STRAP II characters indicating the tails in effect will be appended to the symbol before ADDORD inserts the entire entry in the symbol table. GETCHA makes the entry in the name file which does not remain in memory and is only recalled into memory to be printed on the listing in Pass 2b.

At this time certain information in the unit must also be included in the set up symbol table entry for a symbol referencing done by Pass 2 in establishing the final value. Some of the data to be moved to the symbol table entry are: length of binary output, coded expressions for a P-mode data description on a still "ambiguous" op, explicit data description information of a DD or the coded expression for its symbolic data description, and coded expression for a dimension of a DR. (See Appendix A.)

Note the set up symbol table entry for a SYN or a MCP op has already been made in their respective pseudo-op subroutines.

A special (or "phoney") symbol table entry is set up for a pseudo op which should not have a name but yet does have one. A named pseudo op which should not have a name is not guaranteed to have a full or half word address when referenced.

If ADDORD of table management informs MAIN that the entry just requested to be put in the symbol table is multiply-defined, MAIN has the subroutine MULTI adjust the symbol table entry of the multiply-defined symbol.

Box 54

A set of six error flags is OR'ed into each unit after a unit is processed in Pass 1 and after a unit is processed in Pass 2a. (During each pass processing there is an intermediate error flag buffer where the occurrence of the errors is recorded as detected.) The first five flags denote whether a symbol is undefined, multiply-defined without contradictions, circularly defined or defined in terms of an error symbol. The sixth flag denotes whether the suppress error message, (S), appeared before the instruction. At the end of Pass 1, only the sixth flag could be on. Later in Pass 2 when VALUE is evaluating a coded expression which contains error symbol(s), VALUE informs the using program which in turn sets on the appropriate error flag(s). UITER, too, does some detecting. However, while the undefined, circularly defined, and contagiously defined symbols are detected

after Pass 1, the multiply-defined type symbols are detected and recorded by MULTI in the symbol table during Pass 1.

Boxes 57-61

The unit is now given to INTOUT to be put out on an external storage unit. The counter of the units is incremented. (During the trouble-shooting of the Pass 1 processing, the contents of this counter is available in the general work area as an indication of which unit is being processed. The counter is also available in Pass 2 to give the total number of units in the assembly.) If this is not the unit for the END pseudo-op, the Pass 1 procedure repeats itself and begins processing the next instruction at MAIN. Now if Pass 1 has just received the semicolon character from GETCHA while processing the END unit, Pass 1 will have the file of units terminated by going to the INTOUT subroutine with a special calling sequence. After Pass 1 has processed the unit for the END pseudo-op, INTOUT terminates the file of units with an index word of particular format. This end-of-file indication enables INTIN to know when the last unit has been processed in Pass 2a and in Pass 2b. The final action taken by Pass 1 is to enter the symbol $00 in the symbol table.

Box 62

The between-passes procedure that occurs after Pass 1 includes the following actions:

1. The between-passes procedure has MCP bring in all of STRAP II's Pass 2 from disk.
2. The between-passes procedure turns off the internal Pass 1 indicator, Fpass1, and turns on its Pass 2 counterpart, Fpass2.
3. The between-passes procedure writes the current pass indication on the numeric display lights of the console.
4. The between-passes procedure now enters UITER for the iteration of the symbol table.

Boxes 63-66C

The type of processing for DD's during Pass 1, of course, depends on whether the DD is alphabetic or numeric. (Even though a DDI unit is marked as such, the unit of a DDI receives the same processing as that of a DD during Pass 1.) Since the output of a DD is variable in length, the binary output for a DD is not built up in the usual skeleton op area in the fixed format of the unit. Instead the binary output will be built up in the currently available free area in the unit. The current setting of Ibuffw is saved in the fixed format field at Ziotpt. 32 for the later location of the binary output. If it is an alphabetic DD, the Pass 1 processing is rather straightforward. After some indicator settings are fixed by MQDD, MQDALF handles the alphabetic DD's. The comma is in the value field of $3 when MQDD goes to MQDALF. After completing the conversion MQDALF returns to main flow of Pass 1 with the value field of $3 containing the next non-blank character after the terminating character of the DD. MQDALF converts the characters of the alphabetic DD output code and puts the results in the next available area in the variable length portion of the unit. The length and character count will also be inserted in the unit. The byte size is significant in the assembling of an alphabetic DD in codes: IQS, A8, and A6. If an absolute byte size of less than eight had been specified, MQDALF truncates the characters accordingly. On the other hand, if a symbolic byte size had been specified, truncation if any, will be done by OUTPUT in the final pass after the byte size has been evaluated. An end of card block terminates a DD if the terminating character specified in the entry mode was not included in the D field. (Also, the value field of $3 still contains the false semicolon when MQDALF returns to MAIN.) While MQDALF is requesting the A8 characters from GETCHA, MQDALF has the bit indicator, Gcaz on so that GETCHA will give the input blanks to MQDALF.

Now for the numeric DD's, there are three subroutines responsible for their processing in Pass 1. These subroutines are MQDD, MQDNUM, and DNUM. MQDD has the responsibility of taking care of the radix, which may be 2-16 in STRAP II, the entry mode, multiple numeric cases, the special sign (S), the special exponent (X), and the regular exponent (E). DNUM converts each operand in the statement (after collecting the characters from GETCHA) to a double precision floating point number, and MQDNUM evaluates the field to the semicolon, comma, or comment character.

Note that during Pass 1 the binary output in the unit for a numeric DD is in a floating point form. For all numeric types of DD's, except for Fn-type, the form is that of a double precision normalized floating point number. For the Fn-type numeric DD, the form is that of a double precision unnormalized floating point number. (See Appendix A.) Therefore, OUTPUT must always adjust the binary output in the unit of numeric DD to the specified mode, etc., in order to obtain the final binary output.

Box 67

Using the pseudo-op number and a digit-select-type branch technique, MAIN now determines what action must be taken by Pass 1 on the units of the individual pseudo-ops. Note the pseudo-op term was formerly applied to an instruction that could not be executed during the run of a program. However, during STRAP II

processing, any code that produces a single piece of binary output of predetermined length is not considered a pseudo-op. Thus the operations such as XW, CF, RF, VF, CW, INDMK, and CNOP are not in the pseudo-op branch table. (See Appendix B.)

For many of the pseudo-ops, all that MAIN has yet to do before performing the end-of-instruction procedure is to collect the statement characters. This is accomplished by a tight reloop into GETCHA until GETCHA "kicks out" a semicolon character. The pseudo-ops are: PUNFUL, PUNNOR, PUNORG, PRNS, PRND, PRINID, SKIP, NOPUN, NOPRNT, PRNOR, PUNALL, SPNUS, PRNTALL, NOSEQ, and RESEQ.

DUPLI: For a DUPLI, MDUP saves the pseudo-op's parameters, n and d. The index word Xrepl, in positions 0-17 will contain the number (n) of the input cards to be duplicated; the count field, Xrep2, will contain the number (d) of duplications for each of the n input cards. MDUP also turns on the indicator, Xsp3, to tell XINPUT that it is in the duplicating mode.

XINPUT will stack each of the next n card images in a special memory block before giving each of these n card images to GETCHA. After XINPUT has stacked n card images, then ZINPUT gets the next n times d card images from its own duplicating block rather than from the regular input source. XINPUT resumes fetching card images from the regular source after it has cycled through its duplicating block d times.

PUNID: For the PUNID, MPUNID gets the characters from the address field for the ID from GETCHA and stores them in the available work area of the current unit. The value field of Ziotpt in the unit is set to contain the location of the ID in the unit which OUTPUT will later pick up. Blanks are significant in the ID symbol, so MPUNID has the Gcaz indicator on while requesting characters from GETCHA. An ID of less than eight characters is padded with blanks.

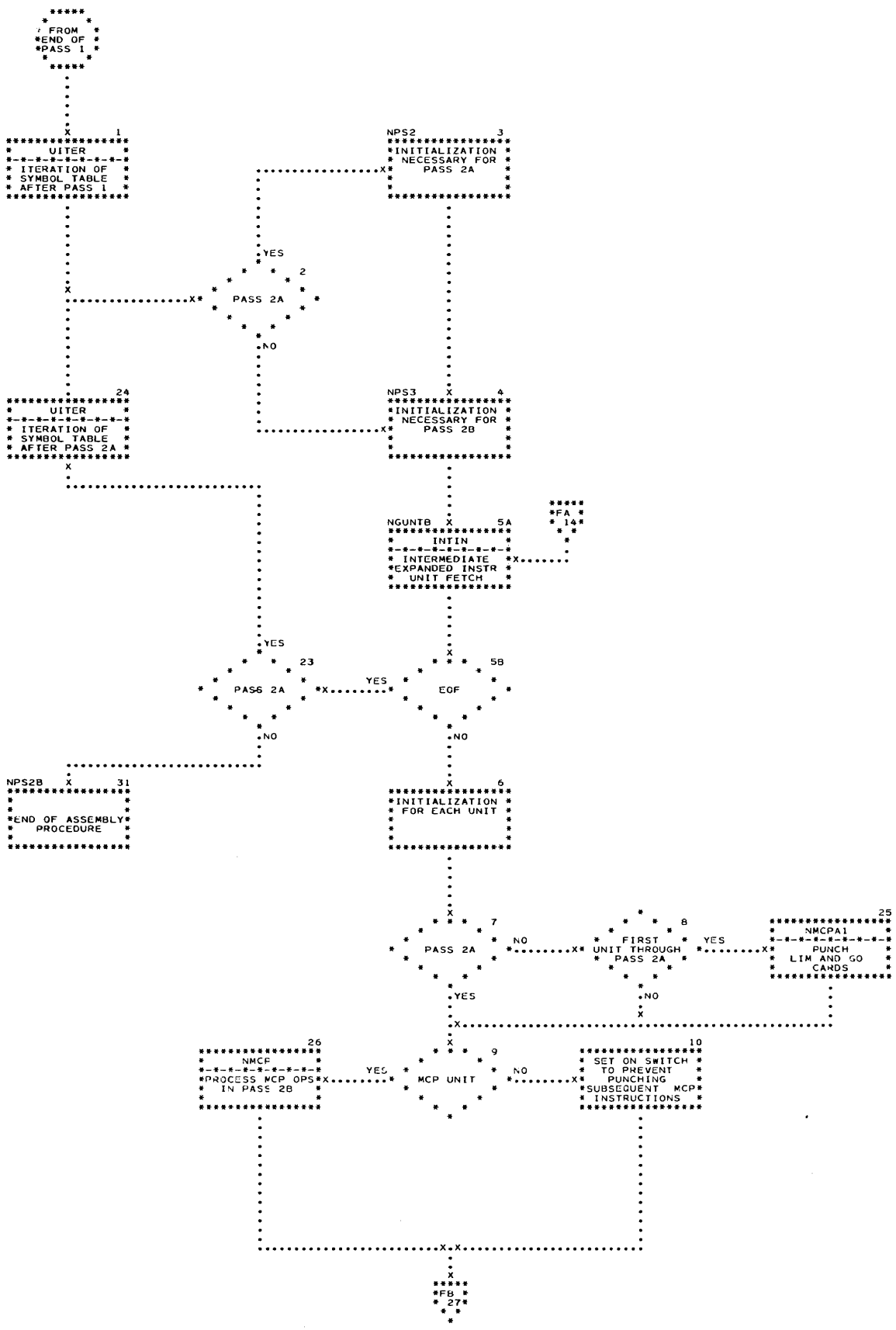
TLB, SLC, END, SLCR: For the pseudo-ops, TLB, SLC, END, and SLCR, Pass 1 has the Pass 2 indicators set on and then has the statement field of the op encoded by GETFLD. Note since there is not a set of INSERT index words for these pseudo-ops, this subroutine sets up here the counter for the number of times to go to GETFLD. (The END instruction must be the only instruction in its card block whenever STRAP II is running with the indicator, Hhcom4, on.)

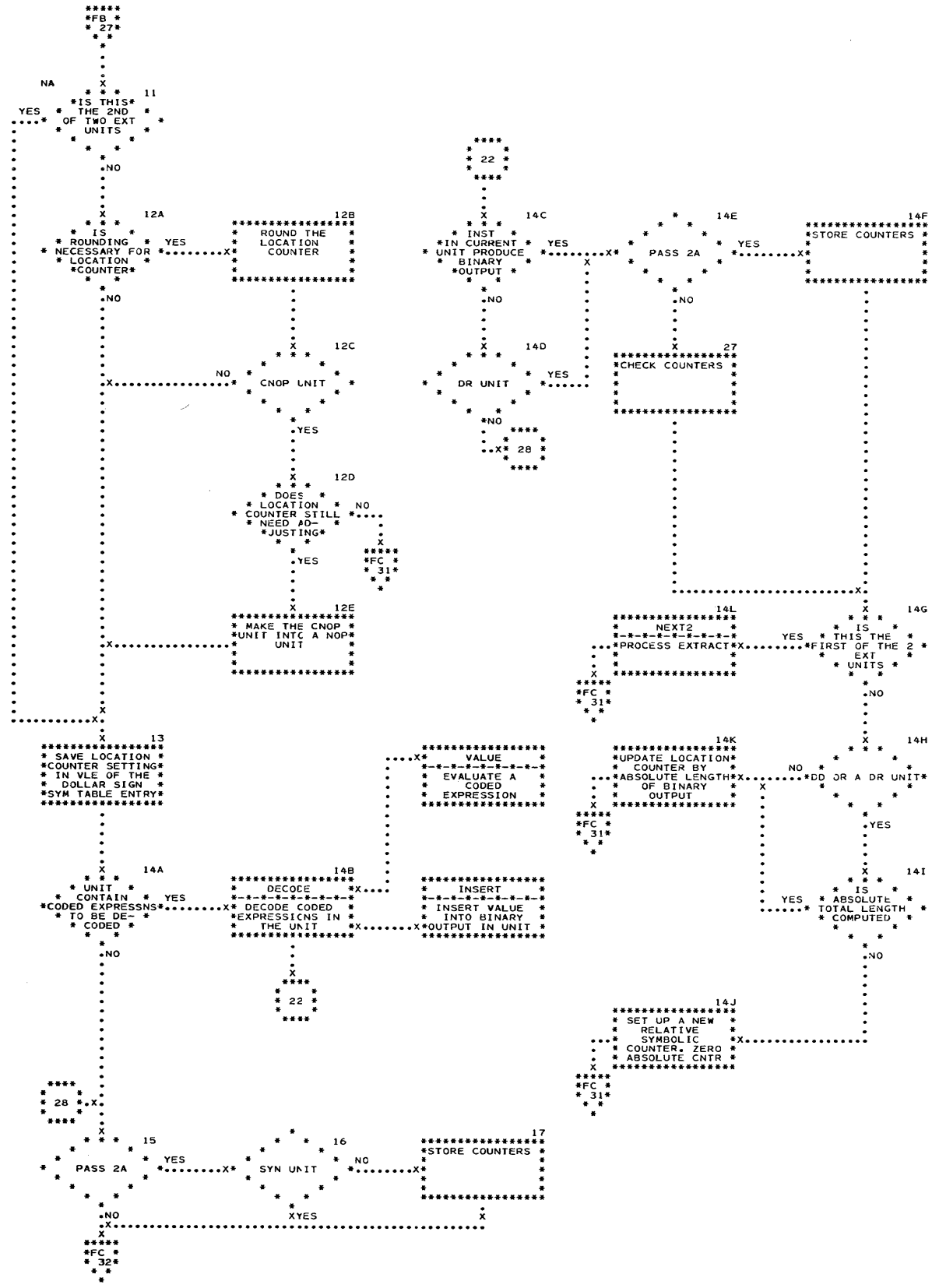
EXT: For the pseudo-op, EXT, STRAP II forms two internal processing units. The EXTract procedure in MAIN, MX, builds up the first unit with the coded expressions of the parameters. After MX has returned to main flow, MAIN will form another unit for the material being extracted upon.

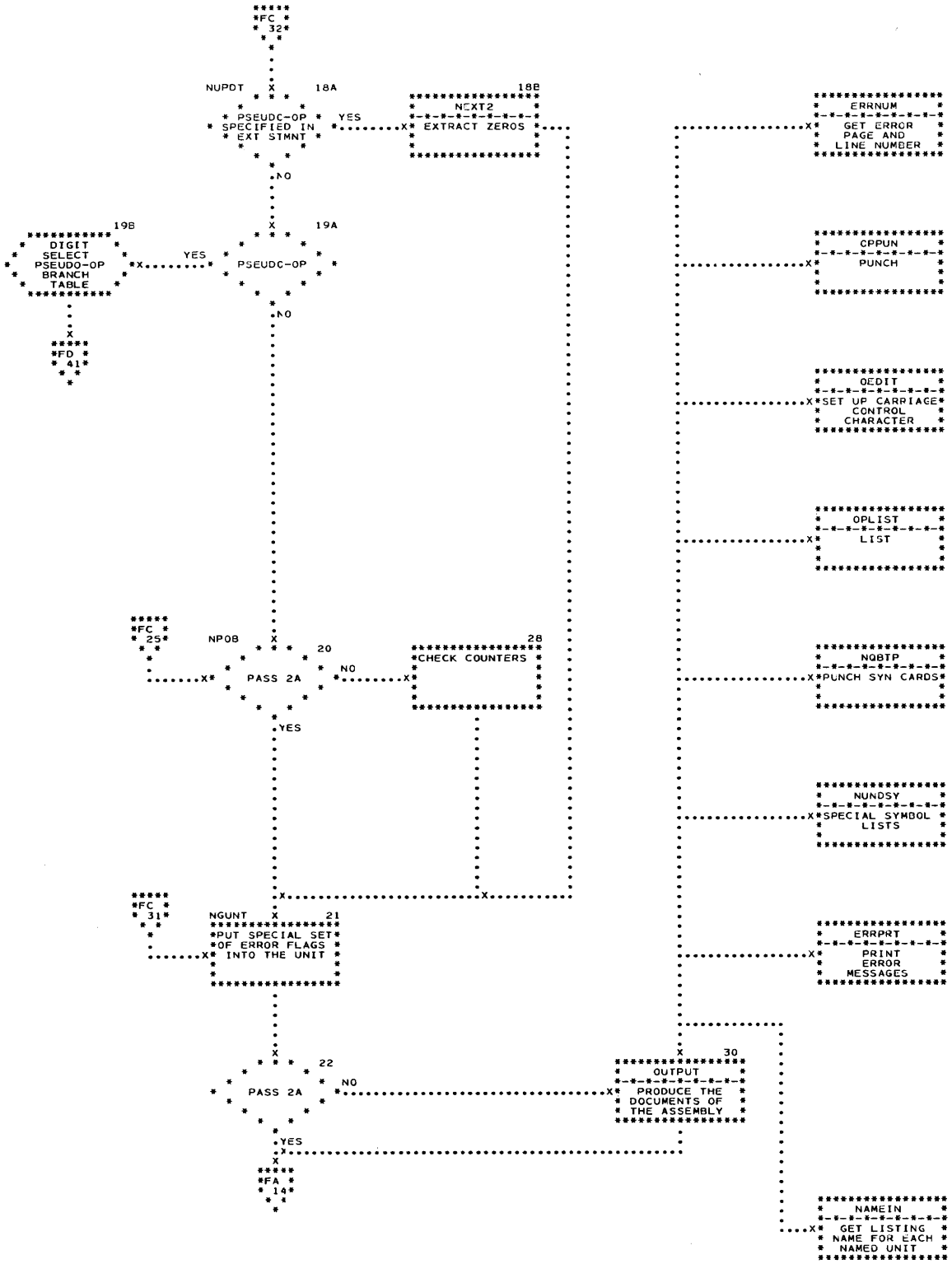
SYN: The main purpose of MQS, which processes each SYN in Pass 1, is to build up the symbol entry in the Msyte area. (The symbol is actually entered in the symbol table at the end of the card block.) The dds information of the SYN op is transferred from the unit to the Msyte area. If any coded expression exits for an undetermined byte size and/or field length, it is moved to the Msyte area from the unit. Since the only coded expressions that could be in the variable part of the unit now are those for the dds, and these are now in Msyte, the available area in the unit buffer is redefined to be the first bit after the fixed portion of the current unit. Since MQS wants the coded expression to be made by GETFLD for the single address field of the SYN op put in Msyte, the upper boundary location of the buffer for the encoding has to be changed because the encoding is not going into the unit. The previous setting of Bcfxtx does not have to be saved because no more coded expressions will go into this unit for this instruction. Besides changing the upper buffer limit, an indicator has to be turned on to tell GETFLD that the encoding is not being put in the usual unit area because GFLONG is only set up to handle the overflow of the unit buffer area, but not the overflow of Msyte.

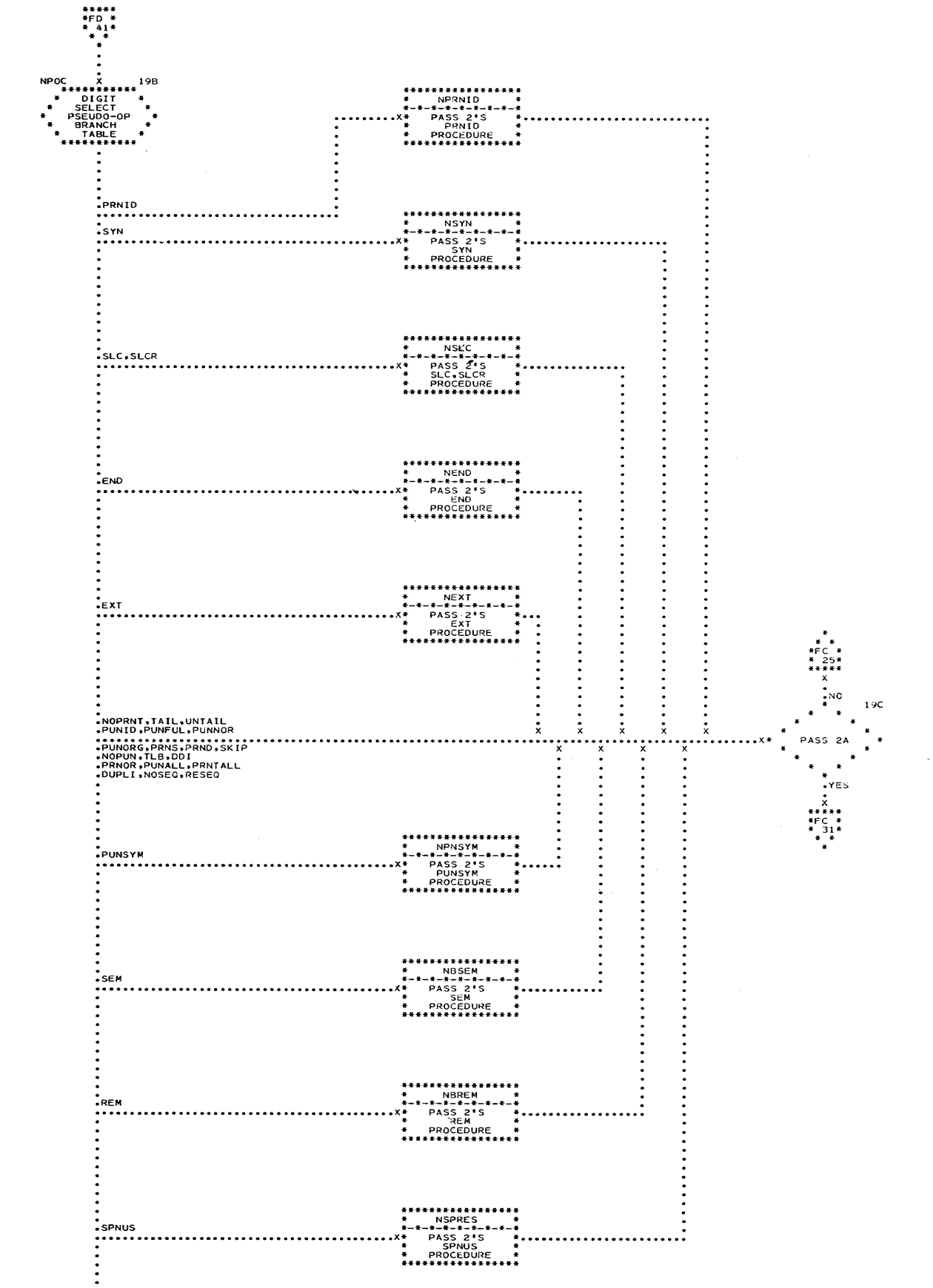
STRAP II main flow knows that there is only one address field to be considered on a SYN unit, but the error message routine does not; the error message field counter must be incremented. The Zifall indicator is turned off to indicate to Pass 2 that there are coded expressions associated with this instruction which are not yet evaluated. Zifp2 is not turned on because the coded expressions will get directly decoded in UITER through VALUE rather than in Pass 2 through DECODE. If there is a dimension in the address field, MQS uses the MDIMRT subroutine to make the separate dimension entry in the vle table. The dimension reference address will be saved in the vle entry of the SYN. (See Appendix A.)

During MQR which handles the DR pseudo-op, the presence of a null dds will cause the dds, (N, 64, 4) to be assigned to the unit of the DR. If the address field is not null, MQR has the basic dimension entry made by MDIMRF. This entry is put in the unit. If the DR has a name, the basic dimension entry prefixed by the appropriate header bits will also be put later into the vle table. (See Appendix A.) The location of the basic portion of the vle dimension entry in the unit is saved in Ziotpt field of the unit. Before returning to main flow to complete the processing of the statement, MQR has VDIMEN try to evaluate the products of the dimensions. The intermediary products of the dimensions are kept in the dimension entry. When the final product is available (now possibly or later in Pass 2), it is put as a negative number into the value field of Ziotpt. The negative


FROM END OF PASS 1
1
UITER
ITERATION OF SYMBOL TABLE AFTER PASS 1
NPS2
3
INITIALIZATION NECESSARY FOR PASS 2A
YES
2
PASS 2A
NO
24
UITER
ITERATION OF SYMBOL TABLE AFTER PASS 2A
NPS3
4
INITIALIZATION NECESSARY FOR PASS 2B
NGUNTB
5A
INTIN
INTERMEDIATE EXPANDED INSTR UNIT FETCH
FA
14
YES
23
PASS 2A
YES
5B
EOF
NO
NPS2B
31
END OF ASSEMBLY PROCEDURE
6
INITIALIZATION FOR EACH UNIT
7
PASS 2A
NO
8
FIRST UNIT THROUGH PASS 2A
YES
25
NMCPA1
PUNCH LIM AND GO CARDS
YES
NO
26
NMCE
PROCESS MCP OPS IN PASS 2B
YES
9
MCP UNIT
NO
10
SET ON SWITCH TO PREVENT PUNCHING SUBSEQUENT MCP INSTRUCTIONS
FB
27


FB 27
11
IS THIS THE 2ND OF TWO EXT UNITS
NA
YES
NO
12A
IS ROUNDING NECESSARY FOR LOCATION COUNTER
YES
NO
12B
ROUND THE LOCATION COUNTER
12C
CNOP UNIT
NO
YES
12D
DOES LOCATION COUNTER STILL NEED ADJUSTING
NO
YES
FC 31
12E
MAKE THE CNOP UNIT INTO A NOP
UNIT
13
SAVE LOCATION COUNTER SETTING IN VLE OF THE DOLLAR SIGN SYM TABLE ENTRY
14A
UNIT CONTAIN CODED EXPRESSNS TO BE DECODED
YES
NO
14B
DECODE DECODE CODED EXPRESSIONS IN THE UNIT
22
DECODE
EVALUATE A CODED EXPRESSION
INSERT
INSERT VALUE INTO BINARY
OUTPUT IN UNIT
VALUE
EVALUATE A CODED EXPRESSION
15
PASS 2A
YES
NO
16
SYN UNIT
YES
NO
17
STORE COUNTERS
28
FC 32
22
14C
INST IN CURRENT UNIT PRODUCE BINARY OUTPUT
YES
NO
14D
DR UNIT
YES
NO
28
14E
PASS 2A
YES
NO
27
CHECK COUNTERS
14F
STORE COUNTERS
14G
IS THIS THE FIRST OF THE 2 EXT UNITS
YES
NO
14H
ADD OR A DR UNIT
NO
YES
14I
IS ABSOLUTE TOTAL LENGTH COMPUTED
YES
NO
14J
SET UP A NEW RELATIVE SYMBOLIC COUNTER. ZERO ABSOLUTE CNTR
14K
UPDATE LOCATION COUNTER BY ABSOLUTE LENGTH OF BINARY OUTPUT
14L
NEXT2
PROCESS EXTRACT
FC 31


FC 32
NUPDT
18A
PSEUDO-OP SPECIFIED IN EXT STMNT
YES
18B
NEXT2
EXTRACT ZEROS
NO
19B
DIGIT SELECT PSEUDO-OP BRANCH TABLE
YES
19A
PSEUDO-OP
NO
FD 41
FC 25
NP08
20
PASS 2A
NO
28
CHECK COUNTERS
YES
FC 31
NGUNT
21
PUT SPECIAL SET OF ERROR FLAGS INTO THE UNIT
22
PASS 2A
NO
30
OUTPUT
PRODUCE THE DOCUMENTS OF THE ASSEMBLY
YES
FA 14
ERRNUM
GET ERROR PAGE AND LINE NUMBER
CPPUN
PUNCH
OEDIT
SET UP CARRIAGE CONTROL CHARACTER
OPLIST
LIST
NQBTP
PUNCH SYN CARDS
NUNDSY
SPECIAL SYMBOL LISTS
ERRPRT
PRINT ERROR MESSAGES
NAMEIN
GET LISTING NAME FOR EACH NAMED UNIT


FD 41
NPOC
DIGIT SELECT PSEUDO-OP BRANCH TABLE
19B
NPRNID
PASS 2'S PRNID PROCEDURE
PRNID
SYN
NSYN
PASS 2'S SYN PROCEDURE
SLC,SLCR
NSLC
PASS 2'S SLC,SLCR PROCEDURE
END
NEND
PASS 2'S END PROCEDURE
EXT
NEXT
PASS 2'S EXT PROCEDURE
NOPRNT,TAIL,UNTAIL
PUNID,PUNFUL,PUNNOR
PUNORG,PRNS,PRND,SKIP
NOPUN,TLB,DDI
PRNOR,PUNALL,PRNTALL
DUPLI,NOSEQ,RESEQ
FC 25
NO
19C
PASS 2A
YES
FC 31
PUNSYM
NPNSYM
PASS 2'S PUNSYM PROCEDURE
SEM
NBSEM
PASS 2'S SEM PROCEDURE
REM
NBREM
PASS 2'S REM PROCEDURE
SPNUS
NSPRES
PASS 2'S SPNUS PROCEDURE

sign of the Ziotpt value field in a DR unit is the indicator that the dimension has been completely computed.

PUNSYM: The PUNSYM pseudo-op causes MQP to collect the requested symbols from the address field(s) of the PUNSYM through CPLTSY, and then stores each symbol preceded by its character length in the variable section of the PUNSYM unit. The format of the SYN cards punched by STRAP II for the symbols requested by the PUNSYM is included in the description of the NQBTP subroutine. The instruction field counter in the fixed format of the unit and the error message field counter are each incremented for each PUNSYM symbol. The list of symbols will always follow the fixed format of the unit, so the location of the list is easily computed in Pass 2. (location of unit + length of fixed format = location of list.) NPNSYM and NQBTP process the PUNSYM's symbols during Pass 2.

REM, SEM: For REM and SEM processing, a four word intermediate error message mask is set up in the unit of the REM or SEM; each bit in the mask will correspond to the status of the corresponding numbered error message (0=restore, 1=suppress). As a result of the current instruction, the major error mask used by the error message routine is updated by the current intermediate mask whenever each SEM or REM unit is being processed. The location of the intermediate message mask in the variable part of the unit is in the value field at Ziotpt. The SEM-REM subroutine in Pass 1 uses GETFLD and VALUE to establish the number of each message in absolute form. There will be no decoding necessary for this unit in Pass 2.

TAIL, UNTAIL: To handle a tail, MQTAIL collects the characters of the tail, sets up a tail entry, and enters it into the tail table. (See Appendix A.) Note that the tails are not in the regular symbol table and thus the tail table has its own control block for the table management subroutines of TABMAN. While a TAIL pseudo-op is in effect, any subsequent symbol appearing in the name field or in an instruction field will be appended with a suffix. A name to be tagged will be treated prior to its entry into the symbol table if a usual NOP instruction had been set to a Branch by MQTAIL. This switch indicates that the name is in the sphere of influence of a TAIL op and thus must be adjusted before being put into the symbol table. CPLTSY, the subroutine which collects the characters of a symbol appearing in an instruction field, appends the STRAP II tailing suffix, when necessary, to the collected symbol before giving the user routine the requested symbol. The UNTAIL pseudo-op is handled by MQT.

LINK: When Pass 1 encounters the LINK pseudo-op in a program being assembled, the MQLINK subroutine is used to put the corresponding coded expression of the instruction -- LVI, 15, $+2 -- into the LINK unit. The location of the op question bits, which is carried both in the unit and in $5, is changed in both places to address the op index of an LVI type instruction. Processing indicators for Pass 2 are turned on in the unit: Zifbo, Zifall, and Zifp2.

## DESCRIPTION OF THE ACTION INDICATED ON THE FLOW CHART OF PASS 2

### Box 1

A program using STRAP II can have instructions containing symbols in the data description and/or address fields, which symbols have not been previously defined in the program. Therefore, the value of each symbol is not available at the end of Pass 1. Thus, the establishing of values for the binary output and the settings of the location counter must be done during a two-part Pass 2.

UITER inspects the symbol table twice, once between the end of Pass 1 and the beginning of Pass 2a, and again between the end of Pass 2a and the beginning of Pass 2b.

After Pass 1, UITER iterates over the symbol table to establish a data description and value for each symbol. Each symbol table entry is examined and any symbol which is not complete, but still needed to evaluate the original symbol, is kept in a push down list and in turn evaluated in an attempt to complete both the data description and the value of the original symbol. A non-P mode symbol initiates a new level. As UITER analyzes each entry in the symbol table, UITER updates the symbol counter for the error message procedure which may be used during UITER.

UITER plays the major role in evaluating each SYN instruction. Only UITER can detect the undefined symbols in a SYN chain. During the first scan of the symbol table, UITER marks these entries of the undefined symbols. Furthermore, while completing each symbol table entry in the first scan, UITER also turns on the appropriate symbol error flags in the symbol table entry if a symbol is either circularly defined or contagiously defined. UITER assigns a value of zero to the base symbol in a circularly defined symbol chain.

The general detection of error symbols occurs so: After Pass 1, UITER turns on an indicator to inform VALUE that it can now make an undefined-type entry in the symbol table during each UITER or Pass 2 evaluation of any symbol which VALUE can not find in the symbol table. (When Pass 1

used VALUE to evaluate a coded expression formed by GETFLD, it was allowable for VALUE not to find the symbol in the symbol table since the symbol table was incomplete during Pass 1. But at the end of Pass 1, the symbol table contains an entry for each name that has appeared in the name field of a card block.) Note that all the regular symbol table entries have been made before Pass 2a, but Pass 2a does make entries in the symbol table for the undefined symbols appearing in non-SYN address fields, for the location counter values that are candidates for hi-lo values, and for the STRAP II symbolic counter symbols that are needed until absolute increments are known. Whenever VALUE is asked to evaluate a symbol, VALUE marks the symbol as having been referenced from an address field. The multiply-defined symbols are detected in ADDORD and are resolved and marked in MULTI during Pass 1. On the final scanning of the symbol table, NUNDSY can easily recognize the error symbols marked for special listing. Of the five different type of irregular symbols, only the multiply defined and the multiply defined with contraditions are mutually exclusive.

Boxes 3, 4

Because of the overlapping functions of Pass 2a and Pass 2b, there are unique indicators associated with each of the phases of Pass 2. The Pass 2a indicator, Np2ind, is set on now. It will be turned off between Pass 2a and Pass 2b.

Pass 2a goes to INTIN an additional time after receiving the END unit, so INTIN by its end-of-file condition thus obtained, can set up the recycling through the units during Pass 2b. The entry into the between passes procedure is made from INTIN. This re-entry location is set now.

Also included in this initialization process is the saving of the location of the ($) symbol table entry since the setting of the location counter for each instruction must be put into this entry. The subroutine for bringing in the units into internal storage and the error message subroutines are initialized. Miscellaneous indicators are set. The unit counter of the error message routine and $4 and $6 are set to zero. The absolute counter is set to its initial $41._8$ location (0 if in PUNREL mode). This initial setting of the absolute counter is entered into the hi-lo table now as the first entry. If the first unit is that of an SLC pseudo op, this low entry will be overruled.

Boxes 5A - 6

For each new unit INTIN brings into process, the following takes place at this point: The field counter for field identification in the error message is zeroed; the unit counter is incremented and the symbol counter is updated. (The field counter is incremented during the Pass 2 procedures which process address fields, e.g., generally by DECODE.) $3 is set up with the location of the new unit. $5 is set up with the location of the op index for the new instruction. The five error flags for an erroneous symbol are zeroed. If the unit has a name associated with it, the location of the function portion of the vle part of the symbol table is computed and saved.

Boxes 9, 10, 26

Master control program's units are processed during Pass 2b. At this point in Pass 2a, the first op that is not an MCP op (comments excepted) causes a switch to be set which will prevent non-contiguous MCP ops from being punched in Pass 2b, although they will be listed.

Box 11

During the processing of EXTract units, the location counter is adjusted only for the first of the two units.

Boxes 12A, 12B

Since the processing of a previous instruction only incremented the location counter by the length of the binary output, rounding tests must be made at the beginning of processing of each new unit in Pass 2a. If the unit is one for a DD or for a DR in a non-floating point mode, there is no rounding necessary. Either of these type of units is sent on to the next processing step. If not, tests are made by using the op question bits to see if the instruction in the unit must have a full or half word address. Once the type of op has been established, the counter can be rounded, if necessary, immediately if $6 is zero, i.e., if there is no symbolic portion of the location counter.

If however there is a symbolic counter portion, changes may have to be made on the symbolic counter. If the present symbolic counter symbol is at the required type location, then only the absolute counter is adjusted for any necessary rounding. Otherwise a new symbolic counter symbol is entered into the symbol table marked with the appropriate full or half word indicator; $4 will be set to zero again; and $6 will contain the location of the new STRAP symbolic counter symbol.

Boxes 12C - 12E

Further adjustment to the location counter may be necessary if the present unit is a CNOP. The

CNOP unit will be effectively processed as a NOP unit if the adjustment is necessary.

Box 13

The current setting of the absolute and symbolic portion of the location counter are put in the function portion of the symbol table entry for the ($)

Boxes 14A - 14L

Here in Pass 2a the unit which contains any coded expressions that have to be decoded is sent to DECODE. (Any unit which produces binary output — including the DR, DRZ, EXT, LINK, DD and DDI ops — will necessarily go to DECODE in Pass 2a.) Note that SYN and MCP-type ops will not be decoded at this point but in their own specialized subroutines in Pass 2.

DECODE uses VALUE in decoding the coded expression(s) in each unit. Included in the information from VALUE is whether the coded expression just evaluated, contained any "error" symbols. If so, DECODE OR's the error symbol indicators set up by VALUE into the intermediate error-symbol mask. During the later processing of the unit in the pass, the intermediate error-symbol mask will be OR'd into the unit so that when the instruction is picked up from the unit to be printed, the instruction will be preceded on the listing by the appropriate error-symbol message.

A fact about DDI processing should be mentioned now. A DDI is processed as a DD during Pass 1. In Pass 2, DECODE will get the DDI evaluated by using special facilities in OUTPUT. Then DECODE adjusts the unit of the DDI so that subsequently Pass 2 will recognize and treat the unit as one for a SYN.

If a unit has just been to DECODE the setting of the location counter (absolute and symbolic counter) is put into the unit and into the corresponding symbol table entry, if any. If the unit produces binary output or is a DR, the location counter must be advanced, also, in preparation for the new unit.

Furthermore, after coming from DECODE, the absolute length of the binary output may or may not be established. If it is, the length is added to the contents of $4 (absolute portion of the location counter). The symbolic error flags will then be OR'd into the unit and the Pass 2a cycle is repeated.

On the other hand, if the op is a DD or a DR, the length of the binary output could very easily still be unestablished. Note that the final length of the binary output for a DD or a DR is kept in Ziotpt field of the unit instead of the usual Zilbo. So a special STRAP II symbol is entered into the symbol table. In the function portion of its vle will be set up a special coded expression in which the sum of the current symbolic counter and the absolute counter is encoded. The location of the vle portion of this new STRAP II symbol replaces the previous contents of the symbolic counter ($6) and zero replaces the previous contents of the absolute counter ($4). Thus a cascading of STRAP II special symbols can be built up to refer to the actual setting of the location counter in a relative way whenever the length of the binary output is not available at update time or whenever the symbolic counter is not at the appropriate type of address at rounding time. Again the final OR'ing of the symbol error flags into the unit is performed and the Pass 2a cycle is repeated.

Boxes 15 - 17

Now to finish the Pass 2a processing of a unit which contains an instruction which is a pseudo-op. The contents of the absolute and symbolic location counters are put into the unit and the vle portion of any companion symbol table entry. This is not done if the unit contains a SYN op.

Boxes 18A, 18B

STRAP II does not prevent extracting from a pseudo-op, but instead supplies zeros and gives an error message.

Box 21

If the op is not a pseudo-op the present setting of the five error flags for an erroneously used symbol is OR'ed into the unit now rather than later and the cycle through Pass 2a is repeated. Note the MCP ops, the DD and the DR ops which have already been treated with follow this action, immediately, too.

Boxes 19A - 19B

The pseudo-op number which was put in the unit of the pseudo-op during Pass 1, is again used in another digit select type branch to determine the specialized treatment of the pseudo-op during Pass 2a. The specific treatment for each pseudo-op that still needs treatment in Pass 2a before the next unit is brought in is described below. The following pseudo-ops only need the wind up procedure in box 20: PUNID, PUNFUL, PUNNOR, PUNORG, PRNS, PRND, SKIP, NOPUN, TLB, DDI, NOPRNT, TAIL, UNTAIL, PRNOR, PUNALL, PRNTALL, DUPLI, NOSEQ and RESEQ.

PRNID: For PRNID, the unit must also be given to the output phase in Pass 2a as well as in Pass 2b. The setting of the error flags are OR'ed into the unit before the output phase gets control. Since the PRNID statements are the only assembly material printed during Pass 2a, all the PRNID's will precede the major listing; the PRNID will also appear in its specified location through the Pass 2b printing.

SLC: For SLC, Pass 2a adjusts the absolute and symbolic counters and makes an entry in the hi-lo table. (The SCLR indicator is turned off, and the hi-lo-entry can-be-made indicator will be set on.)

SLCR: For an SLCR, Pass 2a adjusts the absolute and symbolic counters. No entry is made in the hi-lo table. (The SCLR indicator is turned on and the hi-lo-entry-can-be-made indicator is turned off.)

END: For the END unit Pass 2a enters the location of the END as a possible hi-lo candidate into the hi-lo table.

EXT: For an EXT unit Pass 2a has evaluated the coded expressions for the parameters and saved their values in the unit. The location counter is incremented by the length indicated by the N parameter and the rounding procedure for the next of the EXT units will not be performed. The output phase will pick up the value of the parameter and do the actual extracting from the next unit later in Pass 2b. If the statement of the EXT op is a multiple field DD, then the location counter will be incremented the extra amount.

SYN: For a SYN op, Pass 2a tests if the value is complete; if so, then Pass 2a now performs the action as described in box 21. If the SYN is not completely evaluated at this point, then there is another try to get the value of the SYN by using a section of UITER.

PUNSYM: For each symbol appearing in a PUNSYM pseudo op, the corresponding symbol table entry must be marked for the later punching (to be more exact a bit must be turned on in the function portion of the vle). Since by Pass 2a all the symbols have been entered into the symbol table, the necessary vle's can be marked now during NPUNSYM in Pass 2a. The SYN cards will be punched in Pass 2b during the OUTPUT subroutine through NQBTP.

SEM, REM: For a SEM or a REM the message numbers in the major error mask of the error subroutine are adjusted by the intermediate error mask in the current unit.

SPNUS: For a SPNUS, all that Pass 2a does in turn on an indicator for NUNDSY before performing the end of Pass 2a procedure.

Box 24

If an end-of-file condition occurs when INTIN is fetching the next unit for Pass 2a to process, INTIN will branch to the between passes procedure. (This EOF exit was set up at the beginning of Pass 2a.) Between Pass 2a and Pass 2b, UITER concludes the evaluation of the data description and the value of the symbols in the symbol table. In addition, at this time UITER evaluates the memory bounds of the program being assembled by using the Uhl subroutine which inspects each hi-lo entry in the vle table. On completion of the iteration, UITER returns to Pass 2b with the Pass 2a indicator, Np2ind, off and Pass 2b indicator, Fpass3, on. At this time all the names of the assembly are evaluated.

Again we shall follow the flow through the Pass 2 flow chart. Only this time we shall take all the Pass 2b branches.

By the beginning of Pass 2b all the symbols will have been evaluated in the symbol table. During Pass 2b the floating point format of all numeric DD's will be adjusted and any general parenthetical field entries on the DD's will be treated, the binary output for an EXTract will be extracted, and finally the documents for the program being assembled will be produced.

Boxes 7, 8, 25

On the first cycle through Pass 2b the GO and the LIM cards are produced and punched by NMCPA1. This is the only output that is not punched by OUTPUT. The GO and LIM cards will not appear on the listing (nor will the SKIP and NOPRNT pseudo ops appear on the listing). Note that the GO and LIM cards precede each binary deck produced by STRAP.

Boxes 14E, 27

In Pass 2b, the location counter value previously stored in the unit and in the symbol table entry are

checked against the present setting of the location counter. For a non-binary producing unit the checking is done later. If a discrepancy occurs, an error message will be given and the Pass 2a setting of the location counter will be used.

Boxes 20, 28

During the processing of a unit which does not contain a binary output producing instruction, the checking of the location counter value is done at this point.

Boxes 22, 30

Now before bringing in the next unit, Pass 2b gives OUTPUT control at this point. The function of OUTPUT is to set up the listing and the binary card images which are printed and punched by the I-O subroutines. For the listing of the instruction, OUTPUT gets the location counter, binary output, and symbolic statement including the comment from the corresponding unit. OUTPUT gets the name for each instruction from the name file brought into internal storage by NAMEIN. OUTPUT has the tilting and the page numbering done by OEDIT. ERRNUM sets up and ERRPRT has printed the error messages for OUTPUT. The I-O subroutines, OPLIST and OPPUN, perform the actual printing and punching. If SYN cards have to be made, OUTPUT has the PUNSYM procedure in Pass 2b, the NQBTP subroutine, make them after all the regular binary cards have been produced. NUNDSY is responsible for collecting and having printed those symbols in the symbol table which are undefined, multiply-defined with or without contradictions, circularly defined, or never used in the program. Since OUTPUT uses a digit select type branch technique in determining the appropriate pseudo-op procedure, new pseudo-ops can be added with a minimum alteration to OUTPUT. Note this is the third of the three digit select type pseudo-op tables in STRAP II.

Boxes 19A - 19C

During Pass 2b the following pseudo ops will need no further treatment at this point: PRNID, PUNID, PUNFUL, PUNNOR, PUNORG, PRNS, PRND, SKIP, NOPUN, TLB, DDI, DD, DR, DRZ, NOPRNT, PUNSYM, TAIL, UNTAIL, PRNOR, PUNALL, SPUNS, PRNTALL, DUPLI, NOSEQ, and RESEQ. Their units will go on to box 20 immediately for processing. Their major effect on the Pass 2b process happens in OUTPUT.

SLC, SLCR: For an SLC and SLCR, the location counter is adjusted at this point. The absolute value of the location will be available now. (As with most of the other pseudo-ops, OUTPUT performs a specific action later with a SLC and SLCR. If OUTPUT receives a unit of an SLCR, OUTPUT stops punching cards until the next SLC unit is encountered.)

END: For an END op, the current symbolic and absolute counters are checked and the end of file location for INTIN is now set up.

EXT: For an EXT op, similarly as in Pass 2a, switches are set to prevent rounding and to detect extraction from a psuedo-op in the second of the EXTract's units. The length of the binary output has now been established for the EXT instruction and is saved. Then the counters will be checked in box 28.

SYN: For every SYN, its value is now available in Pass 2b. These bit and integer values are taken from the absolute and symbolic counters of the symbol table and put into the absolute and symbolic counter fields of the corresponding unit of the SYN. Also, the setting of the five error symbol flags that have been accumulated with the symbol are now OR'd into the unit of the SYN. Furthermore, if the symbol has a dimension, the dimension reference address is moved to the unit, too.

SEM, REM: For a SEM or REM unit, the same action takes place now as happened in Pass 2a. Actually the updating of the error mask must take place in each pass of STRAP II since a condition causing an error message to be given can be detected in any pass of STRAP II.

Boxes 5B, 23, 31

The end-of-assembly procedure for STRAP II includes the following actions:

1. The end-of-assembly procedure prints on the listing the total time and the total number of units used.
2. The end-of-assembly procedure prints the message, THE END, on a page by itself, and by using an additional SKIP pseudo-op, displays the message to the operator.
3. The end-of-assembly procedure updates the Communication Record with the information where the STRAP II documents are located.
4. After returning the Communication Record to MCP, the end-of-assembly procedure gives MCP full control to continue the processing.

MAJOR LOGIC AREAS IN THE COMMON SECTION OF STRAP II

| | | |
|---|---|---|
| ERR | ERRor message procedure | Makes an entry in the lists of detected programmer errors. |
| ERRIN | ERRor message INitialization | Initializes the error message procedure before each major part of the assembly by defining the bounds of the current list of detected programmer errors. |
| MOVE | MOVE data | Moves a string of bits from one memory area to another memory area. |
| TABMAN | TABle MANagement | Manipulates the data in the symbol table, in the tail table, and in the branch-on-indicator (BI) table. |
| VALUE | get VALUE of coded expression | Evaluates a coded expression. |
| XERR | input/output ERRor | Is responsible for retrying I-O operations on the disk when a unit check is received by the read/write operation in the user routine. |

SUBROUTINES -- ERRIN, ERR, ERRNUM, and ERRPRT

Details

The cause necessitating an error message can be detected during any of the passes of STRAP II. The cause of an error message can be an instruction which has been illegally specified by the programmer, an instruction which contains a possible cause of error, or a "trouble" condition which has occurred within the assembly process. All the error messages appear at the end of the listing. In general, the detection of each error results in the creation of a 32-bit entry in one of five internal error lists. At the end of the assembly program, each entry in the lists causes a corresponding error message to be printed. Each print out of an error message includes not only the description of the error but also the page and line number of the instruction in question and, as necessary, the address field of the occurring error.

There are four subroutines involved in the recording and printing of the error messages by STRAP II: ERRIN, ERR, ERRNUM, and ERRPRT. The time of their use during STRAP II is described first and then their function. ERRIN is used during the initialization of each of the major parts of STRAP II: Pass 1, Pass 2a, Pass 2b, the UITER between Pass 1 and Pass 2a, and the UITER between Pass 2a and Pass 2b. ERR is used during each end-exception interrupt action that STRAP II forces to occur whenever an error condition is detected within Pass 1, UITER, Pass 2a, UITER, or Pass 2b. ERRNUM is used after each unit has been processed in the OUTPUT routine at a point after all possible errors have been detected in Pass 2b. ERRPRT is used once at the very end of the assembly in Pass 2b.

Since ERRIN is only entered during the initialization of each major part of STRAP II before any error condition can be detected in that part of the assembly, ERRIN's primary function is to stack the starting location (current contents of Errfwa) of each of the five lists being built up by ERR, ERRNUM with page and line information supplied by OUTPUT completes all the error entries in the ERR lists so that the five lists finally can be merged and printed by ERRPRT.

The source of information for ERR are: three counters, Unit (BU, 20), Field (BU, 4), and Symbol (BU, 24); a table at Errtab of ordered 64-bit control words each of which locates a variable length error message; and the bounds, which are contained in Errfwa and Errtop, of each area for the ERR lists containing the 32-bit entries. (See section on serviceability aids for a description of the counters: Unit, Field, and Symbol.)

ERR is responsible for making the 32-bit entry recording the occurrence of an error detected by STRAP II either in the problem program or in the assembly program. If the error is detected while Pass 1, Pass 2a, or Pass 2b is in process, the 32-bit entry has the following form. (Each of these parts of STRAP processes the expanded instruction units in their chronological order.)

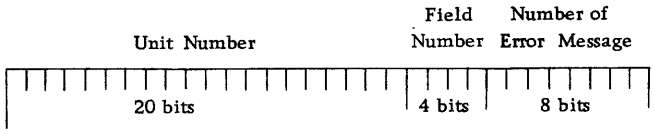



Figure 3. Error Message Record for Pass1, Pass 2a, and Pass 2b

On the other hand, if the error is detected while either use of UITER is in process, the above 32-bit entry has another form. (UITER analyzes the entries in the ordered symbol table.)

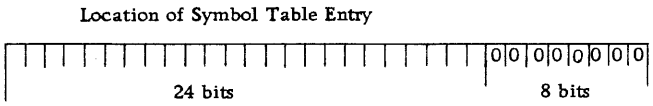



Figure 4. Error Message Record for UITER

Preceding each list, ERR makes the following specialized type entry:

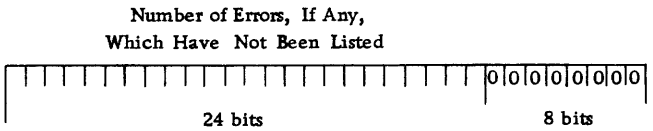



Figure 5. Initial Error Message Record

ERR returns to MCP in order to resume assembly processing.

The number of an error message refers to the rank of the corresponding particular control word in the Errtab table of control words. The method by which ERR establishes the rank of the appropriate control word should be explained. When STRAP II detects an error condition, STRAP II performs an instruction of the type -- EX, A -- where A is the address of a CW instruction which locates the message to be given for the condition detected. The request to execute a CW instruction causes an execute exception interrupt to occur, whereupon MCP takes immediate control. Now the refill field of $15 has the fixed use of pointing to STRAP II's table of exits for maskable interrupts. This table includes the address of ERR as the location MCP is to go to whenever MCP receives an execute exception interrupt from STRAP II. Then after MCP's subsequent entry into the ERR subroutine, ERR manipulates the interrupt location counter value, which MCP has put in the STRAP II table of exits, to get the address, A. Once ERR obtains the location of the control word, its rank in the table of control words is obtained by a simple subtraction. The rank of the CW is used in the 32-bit entry recording the error message.

Serious error messages printed at the end of the listing can also be printed on the console typewriter when the error condition occurs under the installation's optionable use of a Correction card. These serious error messages have their corresponding control words at the beginning of the list of error message control words. The number of these type of control words is given by the quantity Esss to which ERR compares the error message number just calculated.

ERRNUM looks through the five lists for an entry with either a unit number corresponding to the contents of the Unit counter (if an item in the first, third, or fifth list), or a symbol number corresponding to the contents of the Symbol counter (if an item in the second or fourth list). Then using the page and line information supplied by OUTPUT, ERRNUM transforms the error entry into the new format.

The 12-bit page number is made available by OUTPUT in the accumulator at offset 8; the 8-bit line number is in the accumulator at offset 0. (If the found error entry is one from either the second or fourth list, the field designations in the transformed error entry are set to zero.) In addition ERRNUM puts the second and fourth lists in chronological order, the other lists are built up in chronological order originally.

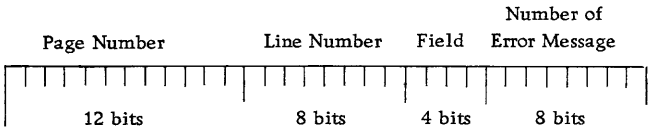
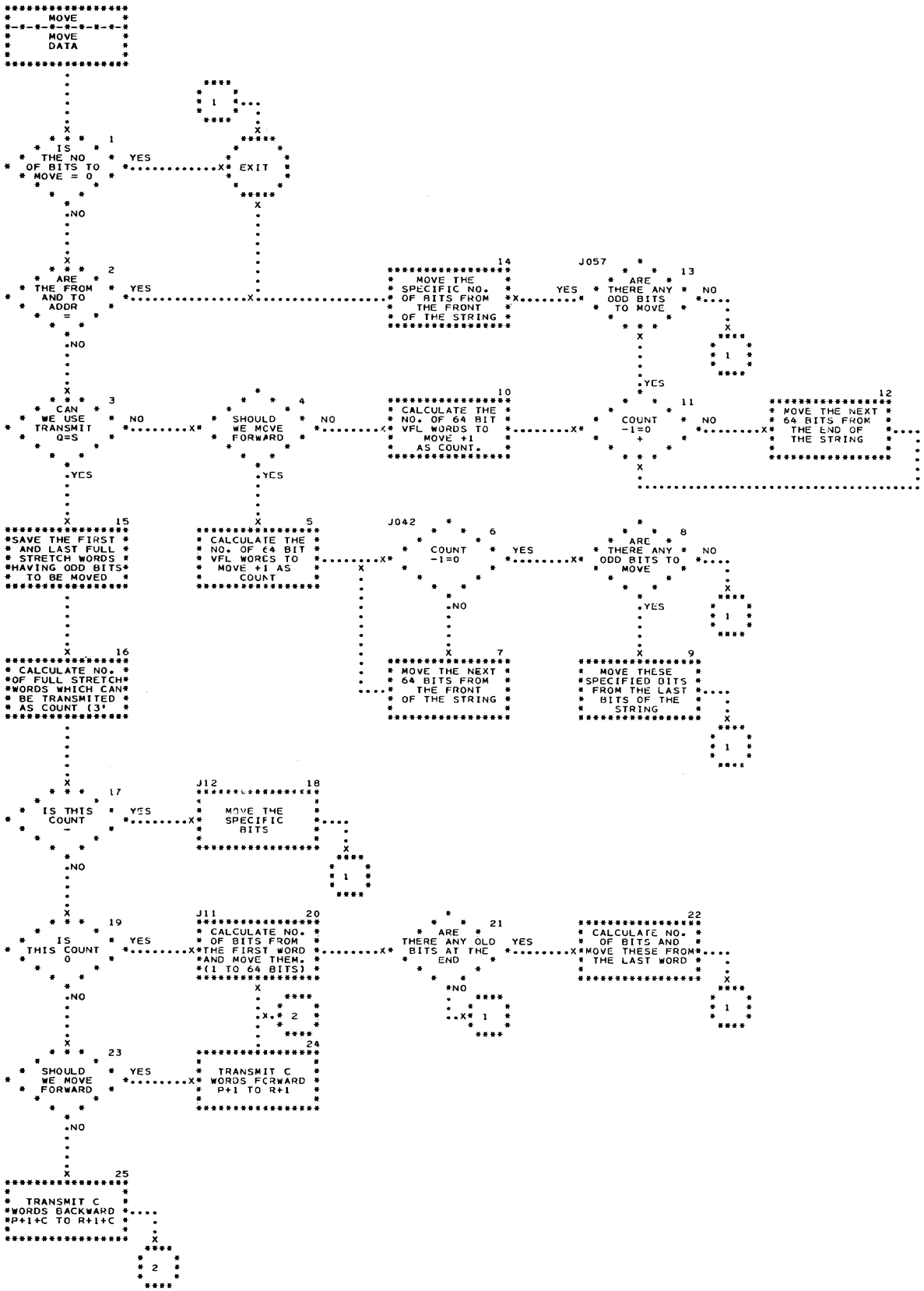



Figure 6. Final Error Message Record for ERRPRT


MOVE DATA
IS THE NO OF BITS TO MOVE = 0
YES
EXIT
NO
ARE THE FROM AND TO ADDR =
YES
NO
MOVE THE SPECIFIC NO. OF BITS FROM THE FRONT OF THE STRING
14
J057
ARE THERE ANY ODD BITS TO MOVE
13
NO
YES
CAN WE USE TRANSMIT Q=S
NO
YES
SHOULD WE MOVE FORWARD
4
NO
YES
CALCULATE THE NO. OF 64 BIT VFL WORDS TO MOVE +1 AS COUNT.
10
COUNT -1=0 +
11
NO
MOVE THE NEXT 64 BITS FROM THE END OF THE STRING
12
SAVE THE FIRST AND LAST FULL STRETCH WORDS HAVING ODD BITS TO BE MOVED
15
CALCULATE THE NO. OF 64 BIT VFL WORDS TO MOVE +1 AS COUNT
5
J042
COUNT -1=0
6
YES
NO
ARE THERE ANY ODD BITS TO MOVE
8
NO
YES
CALCULATE NO. OF FULL STRETCH WORDS WHICH CAN BE TRANSMITED AS COUNT (3'
16
MOVE THE NEXT 64 BITS FROM THE FRONT OF THE STRING
7
MOVE THESE SPECIFIED BITS FROM THE LAST BITS OF THE STRING
9
IS THIS COUNT -
17
YES
NO
J12
MOVE THE SPECIFIC BITS
18
IS THIS COUNT 0
19
YES
NO
J11
CALCULATE NO. OF BITS FROM THE FIRST WORD AND MOVE THEM. (1 TO 64 BITS)
20
ARE THERE ANY OLD BITS AT THE END
21
YES
NO
CALCULATE NO. OF BITS AND MOVE THESE FROM THE LAST WORD
22
SHOULD WE MOVE FORWARD
23
YES
NO
TRANSMIT C WORDS FORWARD P+1 TO R+1
24
TRANSMIT C WORDS BACKWARD P+1+C TO R+1+C
25
1
2

LINKAGE

The calling sequence is:

```
LVI, 15, $+2; B, MOVE
, from address (I)          'referred to as A or P.Q
, to address (I)            'referred to as B or R.S
, number of bits to be      'referred to as N or T.U
  moved (I)
Normal Return
```

RESTRICTION

Currently MOVE destructively uses the accumulator, the arithmetic result indicators, the comparison result indicators, and the result indicators.

USERS

OEDIT, OUTPUT, OPLIST, VALUE, TABMAN

DETAILS

In the description of MOVE's method of moving data, the following nomenclature and examples are referred to:

```
        A = P.Q    = the 'from' address
        B = R.S    = the 'to' address
        N = T.U    = the number of bits to be moved
A + N = Z = X.Y    = the address of the next bit
                     following the data in the
                     'from' buffer
```

A storage word V is used for storing 64 bits of data beginning at P.0. A storage word W is used for storing 64 bits of data beginning at X.0.

The possible situations that can occur are:

1. N = 0 or A = B.
2. Q = S either with A<B or with A>B, and N ≠ 0.
3. Q = S, and N ≠ 0.

FLOW CHART DESCRIPTION

Boxes 1, 2

For situation #1, MOVE does not move any data and returns directly back to the user program.

Boxes 3-14

For situation #3, MOVE, by progressively indexed Load and STore operations, moves T words forward (or backward) from location A (or A + N) to location B (or B + N) if A <B (or if A >B).

Then if U ≠ 0, MOVE picks up the U odd data bits from location A + T (or A) and puts them in location B + T ( or B) if A <B (or if A >B).

Boxes 15, 16

To handle situation #2 MOVE does the following: First, MOVE saves the words at locations P.0 and X.0 in the storage words at locations V and W, respectively, to protect against overlapping.

Then since Q = S and a straight Transmit operation may be able to be used, MOVE computes how many full words are in the data by the formula:

L = N-Y-(64-Q)/64.

Boxes 19-22

If L = 0, as is the case in ex. b, then MOVE moves the last 64-Q bits of the word V to location B. And then if Y ≠ 0, MOVE moves the first Y bits of word W to location B + N - Y.

Boxes 17, 18

If L< 0, as is the case in ex. a, then MOVE moves to location B the N bits which are in the word V at an offset of 64-(Q + N) bits. There is no boundary crossover.

Boxes 23-25

If L>0, as is the case in ex. c, then MOVE transmits forward (or backward) L full words from location P.64 (or P.0 + L) to location R.64 (or R.0) if A < B (or if A > B).

Then MOVE moves the odd number of data bits in the words, V and W, as described above.

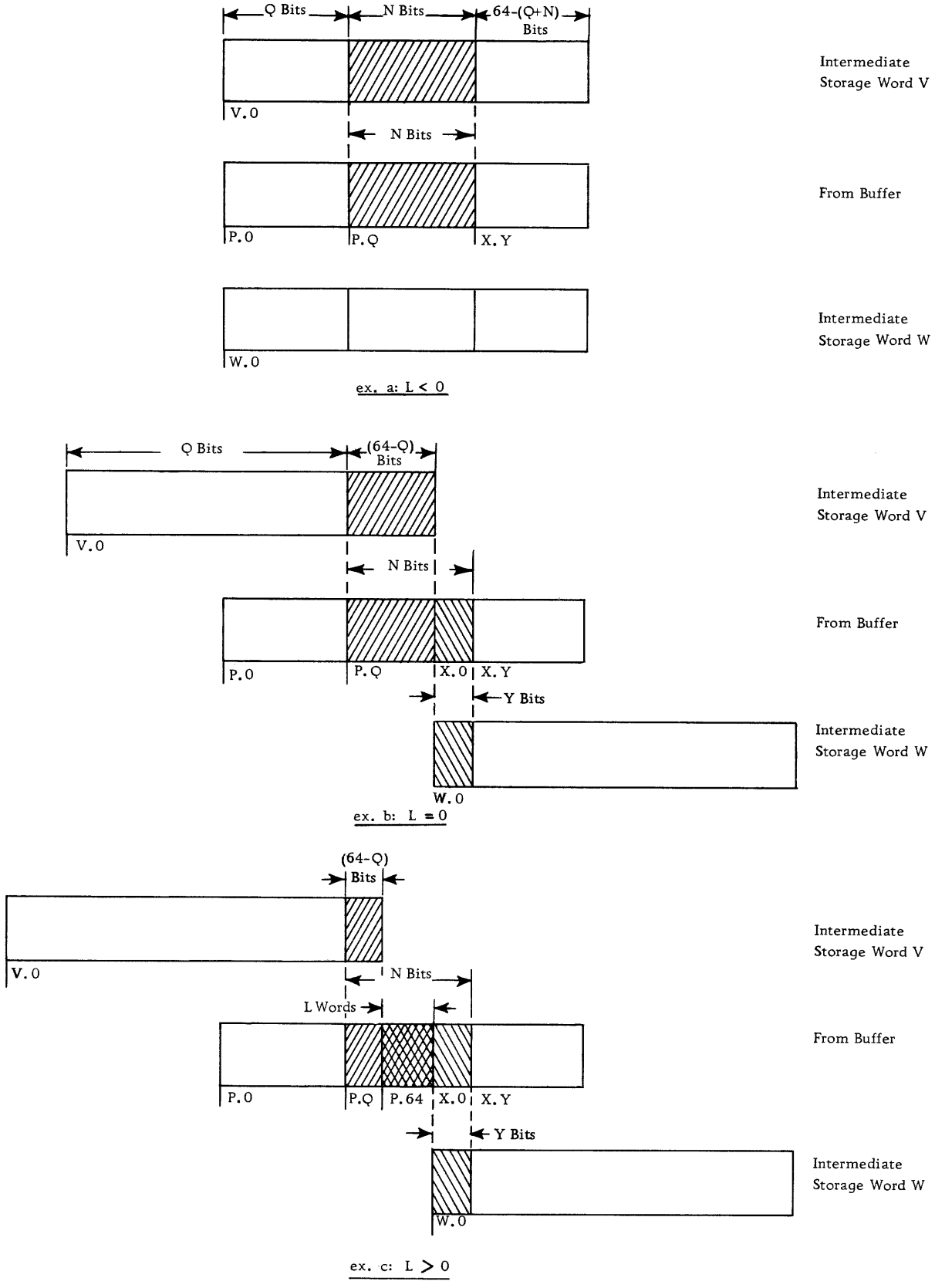
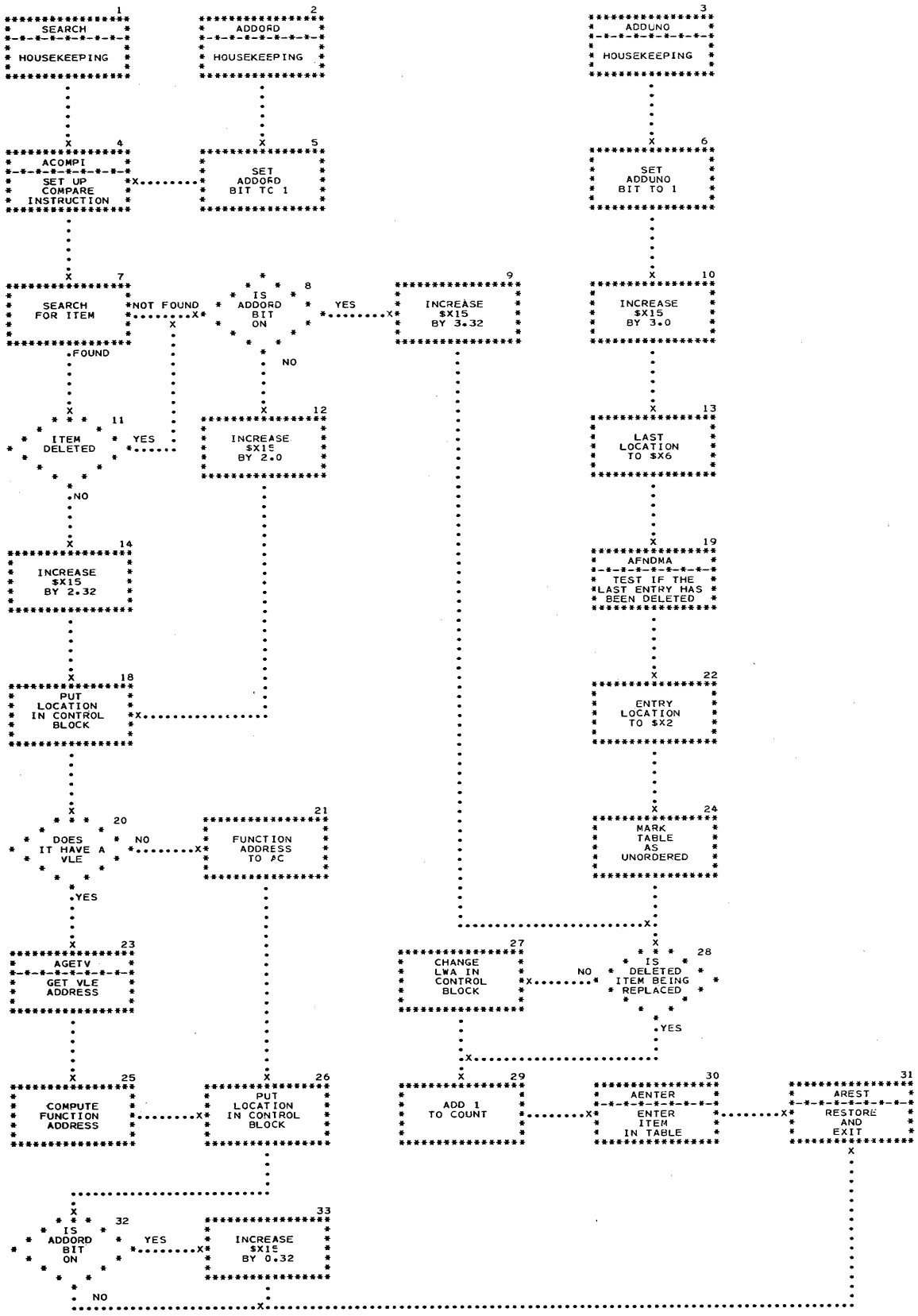



Figure 7. Format of MOVE's Storage Words


1
SEARCH
HOUSEKEEPING
2
ADDORD
HOUSEKEEPING
3
ADDUNO
HOUSEKEEPING
4
ACOMPI
SET UP COMPARE INSTRUCTION
5
SET ADDORD BIT TO 1
6
SET ADDUNO BIT TO 1
7
SEARCH FOR ITEM
NOT FOUND
FOUND
8
IS ADDORD BIT ON
YES
NO
9
INCREASE $X15 BY 3.32
10
INCREASE $X15 BY 3.0
11
ITEM DELETED
YES
NO
12
INCREASE $X15 BY 2.0
13
LAST LOCATION TO $X6
14
INCREASE $X15 BY 2.32
19
AFNDMA
TEST IF THE LAST ENTRY HAS BEEN DELETED
18
PUT LOCATION IN CONTROL BLOCK
22
ENTRY LOCATION TO $X2
20
DOES IT HAVE A VLE
NO
YES
21
FUNCTION ADDRESS TO AC
24
MARK TABLE AS UNORDERED
23
AGETV
GET VLE ADDRESS
27
CHANGE LWA IN CONTROL BLOCK
28
IS DELETED ITEM BEING REPLACED
NO
YES
25
COMPUTE FUNCTION ADDRESS
26
PUT LOCATION IN CONTROL BLOCK
29
ADD 1 TO COUNT
30
AENTER
ENTER ITEM IN TABLE
31
AREST
RESTORE AND EXIT
32
IS ADDORD BIT ON
YES
NO
33
INCREASE $X15 BY 0.32

## DETAILS

The table management routines are a set of programs designed to accomplish the table handling functions of search, ordered and unordered addition, replacement, research, and scan in STRAP II.

Table manipulation is based upon the conformation of design of each table to specific information contained in an associated table control block. The calling sequence for each table management routine includes the location of the table control block and certain individual entrance parameters. Table design and control block information are completely interdependent.

A table entry has a fixed length argument or designator part and a possible variable length portion - the function or use part. All entries of a table consist of at least a fixed length portion, the length of which is specified in the table's control block. (This fixed length portion is also called the dictionary part of the entry.) Whenever an entry either exceeds established fixed length limits or whenever it is desired to split the argument, a variable length entry (vle) is established. If a vle portion exists, its address is contained in the function field of the dictionary entry. A block of storage has been reserved for the consolidated vle portions of the tables. The dictionary part of different tables is located in non-adjacent blocks of storage, but the vle portions of all tables are located in one consolidated vle block of storage.

The control block for each table contains all information pertinent to the fixed length portion of that table. If a vle portion exists, the address is contained in the last 25 bits of the function part of the dictionary entry. All other information concerning the vle is contained in the vle itself. If a vle portion is to be generated, the universal control block (another control block governing all the tables managed by table management) contains the next available location in the consolidated vle table. One characteristic that cannot be too strongly stressed is that a table control block contains defining information about the fixed length entry part of its table only. In addition, the fixed length portion of each individual entry conforms to this defining information. Thus, each table entry has one of three forms:

1. Argument and function in fixed length portion and no vle portion as an entry in the MIBA table.
2. Argument in fixed length portion and function in vle portion as the STRAP special symbol entry in the symbol table.
3. Part of argument in fixed length portion and remainder of argument plus the function in vle portion as an input symbol entry in the symbol table.

The set of table management routines are: SEARCH, ADDORD, ADDUNO, RSERCH, and ANEXT.

Figure 8 presents the format for a table control block associated with either the table for the symbols (taken from the input instructions), or the table for the 7030 indicators, or the table for the tails (taken from the input instructions).

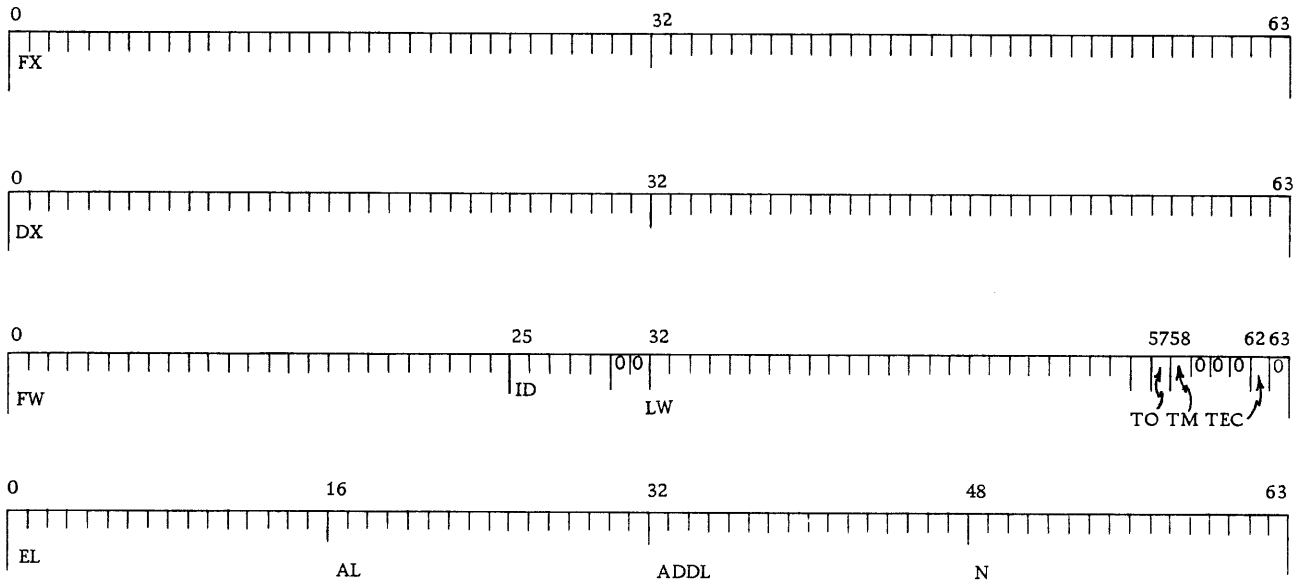



Figure 8. Format for a Table Control Block

Individual Control Block

| Field in Diagram | Contents | Relative Location Symbol | Relative Location | Length in Bits | Restrictions |
|---|---|---|---|---|---|
| FX | Index word referring to function part of an entry | Azfx | 0.0 | 64 | Full word |
| DX | Index word referring to dictionary location | Azdx | 1.0 | 64 | Full word |
| FW | Location of first entry of table | Azfw | 2.0 | 24 | Address of half wd. |
| ID | Identification | Azid | 2.25 | 6 | - - - - |
| LW | Location of last entry of table | Azlw | 2.32 | 24 | Address of half wd. |
| TO | Type of table (0, ordered; 1, unordered) | Azto | 2.57 | 1 | 1 bit |
| TM | Table delete mark | Aztm | 2.58 | 1 | 1 bit |
| TEC | Type of compare | Aztec | 2.62 | 1 | 1 bit |
| EL | Length of fixed entry in dictionary (filled in by subroutine) | Azel | 3.0 | 16 | - - - - |
| AL | Length of argument in dictionary | Azal | 3.16 | 16 | - - - - |
| ADDL | Length of function in dictionary | Azaddl | 3.32 | 16 | - - - - |
| N | Number of undeleted entries in dictionary | Azn | 3.48 | 16 | - - - - |

Universal Control Block

| Field | Length (bits) | Contents |
|---|---|---|
| Auvtab | 24 of half word | Word addressed contains first available bit location of vle table (which is built upwards). |
| Auvbot | 24 of half word | Lowest bit address usable in vle block. |
| Autcb | 18 of half word | Location of first table control block. |
| Aunt | 18 of half word | Number of table control blocks, including any marked for deletion, unless at end. |

FLOW CHART DESCRIPTION

Boxes 1-3

Housekeeping for the three routines accomplished the following:

$0 - $7 are saved.

Azzc and Azzb bits are set to zero.

The location of the table control block is read from the calling sequence and placed in the value field of $5.

Box 4

Acompi sets the field length in the compare instructions for the size of the argument under consideration.

Box 7

If the number of items in the table is less than seven, a linear search is performed. If seven or more, a binary search is used.

Box 14

If the item is found and not deleted, $15 is advanced to return to the found return for SEARCH.

Boxes 18, 20, 21, 23, 25, 26

The location of the equal entry is put into the table control block. If the entry has a variable length entry portion, its location is obtained using Agetv. In both cases the function address is placed in the table control block.

Boxes 32, 33

If the Azzc bit is on, $15 must be increased by half a word to return to the "already there" return.

Box 31

$0 - $7 are restored and exit is made by a ---·
B, -1($15).

Boxes 8, 9

If the item is not found and the Azzc bit is off, $15 is increased to point to the "not found" return. If the Azzc bit is on, $15 is increased to point to the "normal" return for ADDORD.

Boxes 10, 13, 19, 22, 24

Index register 15 is increased to point to the "normal" return. The last entry in the table is placed in $6 and a branch is made to the find delete mark routine. At exit $2 will contain the address of the last entry, if deleted, or the next available location in the table. The table is now marked as unordered.

Boxes 28, 27

If a deleted item is not being replaced it is necessary to increase the last word address in the table control block by one entry length.

Boxes 29-31

The number of entries in the table control block is increased by one and the new entry is moved to the location specified in $2 and exit is made via Arest.

PURPOSE

The main objective of SEARCH is to ascertain whether or not a given piece of information is contained within a table.

LINKAGE

```
LVI, 15, $+2; B, SEARCH
, tbl
, arg (I)
, al (I)
B, a fix-up subroutine          'not found return.
Normal Return
```

DETAILS

The SEARCH objective is accomplished through the use of a main program and several key subroutines. The main program will decide which of two specific table search routines to use, linear or binary. These routines select a table argument, and then branch to a common routine which compares the given and table arguments. If an equal comparison is not made, the specific SEARCH routine will continue to select other table arguments until either an equal table argument is found or the entire table has been processed. Only if the table is ordered and greater than six entries long will a binary search be conducted.

The index words in the control block will contain one of two locations, depending on whether or not an argument in the table equal to the given argument was found. If an equal argument was found, the location of that argument and its function will be in the control table index words. If an equal argument was not found, one of two results may appear in the index words. In the case of an ordered table search, the location of the entry with the next higher argument will be in the index words. For a non-ordered table search, the next location after the table will be given in the index words.

ADDORD

PURPOSE

The objective of ADDORD is to add an entry in its proper place in an ordered table, if the entry is not already in the table.

LINKAGE

```
LVI, 15, $+2; B, ADDORD
, tbl
, arg(I)
, al(I)
, funct(I)
, fl(I)
B, a fix-up subroutine        'Already there exit.
B, a fix-up subroutine        'Error exit, unordered
                                table, item not in
                                order.
Normal Return
```

RESTRICTIONS

1. The table must be in ascending order.
2. It is the programmer's responsibility to make certain that adding an entry will not overflow the table and possibly overlap other information or tables.

DETAILS

An initial check is made to see if the table is ordered. It not, the entry will be added out of order. If the table is ordered, the location of a table argument equal to or next higher than the given argument is obtained by use of the SEARCH routine. If no equal argument is found, the table will be moved up the correct amount to permit insertion of the new entry in order. All this will be done in conformation to the control block specifications. The control block will then be updated.

On return to the 'already there' exit, the index words in the control block will locate the equal item. If a 'normal return' is made, the control block will reflect the change in the number of entries in the table, and the location of the properly inserted entry. The 'error return' exit will be made from the ADDUNO routine if an unordered table had been detected, and the entry will be added to the end of the table.

PURPOSE

The objective of ADDUNO is to add an item to a table.

LINKAGE

```
LVI, 15, $+2; B, ADDUNO
, tbl
, arg(I)
, al(I)
, funct(I)
, fl(I)
Normal Return
```

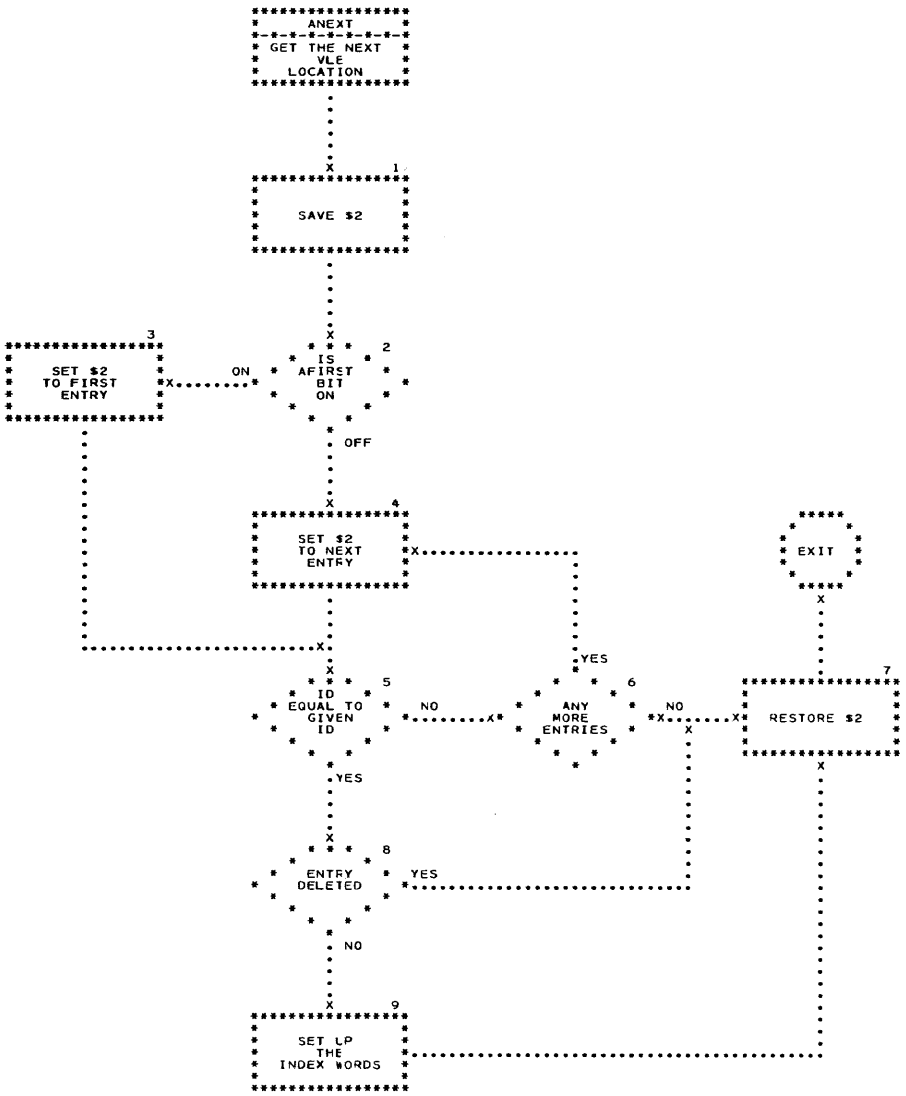
RESTRICTIONS

When the given item is added to the table, this routine does not check for available space. Therefore, the beginning of the next table may unwittingly be destroyed. The overall bookkeeping of all tables is left to the programmer, and it is the programmer's responsibility to avoid obliterating his own memory.

DETAILS

The routine checks if the last entry in the table is marked for deletion. If it is, it is replaced by the given item. If not, the given item is placed immediately after the last entry of the table. This routine enters the contents of location arg(I) and funct(I) into the table, adjusts the address of the last vle entry in the universal control block when necessary, and makes the following adjustments to the table control block to correspond to the new status of the table:

1. The count of entries (Azn) is advanced by one.
2. The table is marked unordered.
3. The location of the last dictionary entry (Azew) is changed when the given item is added at the end of the table.


ANEXT
GET THE NEXT VLE LOCATION
1
SAVE $2
2
IS AFIRST BIT ON
ON
OFF
3
SET $2 TO FIRST ENTRY
4
SET $2 TO NEXT ENTRY
EXIT
5
ID EQUAL TO GIVEN ID
NO
YES
6
ANY MORE ENTRIES
NO
YES
7
RESTORE $2
8
ENTRY DELETED
YES
NO
9
SET UP THE INDEX WORDS

## PURPOSE

The objective of ANEXT is to scan the variable length entry table for entries associated with a particular table control block (i.e., for those entries with the same ID).

## LINKAGE

```
LVI, 15, $+2; B, ANEXT
, ID                                  'See restriction.
, linkage to a fix-up subroutine      'See restriction.
,                                     No more vle
                                      return.
Normal Return
```

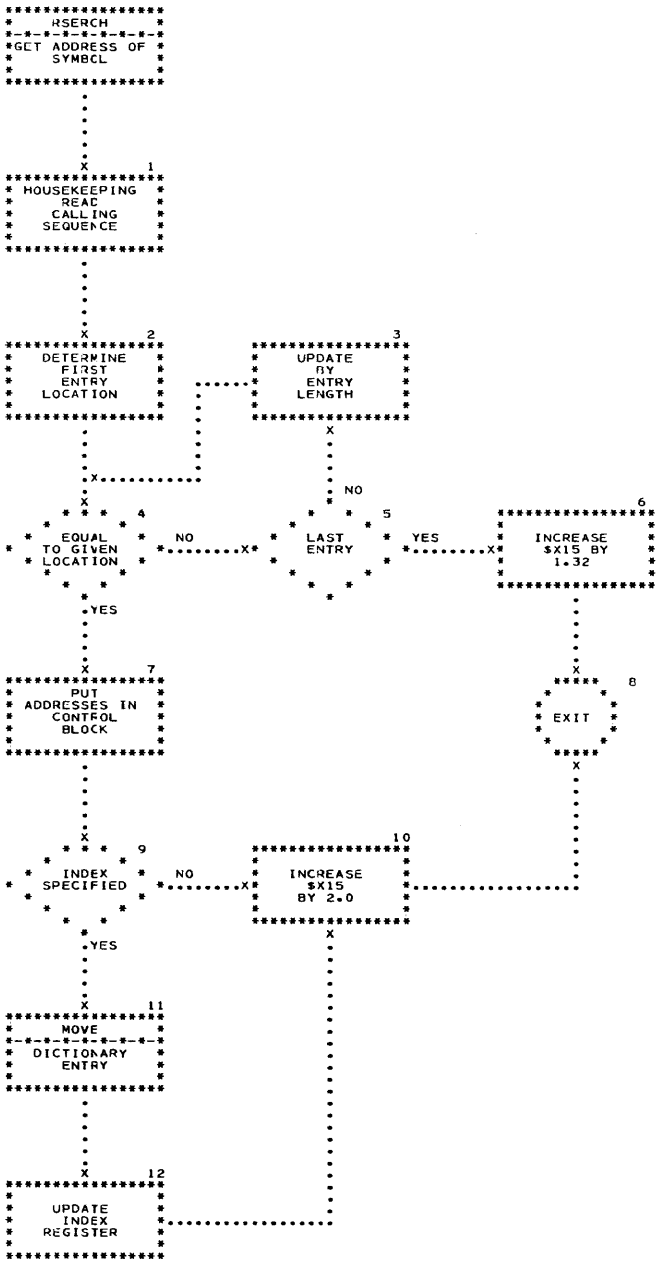
## RESTRICTIONS

Initially the bit, Afirst, must be on. ANEXT turns the indicator off. In fact, whenever it is desired to restart the scan at the beginning of the vle table, Afirst should be turned on.

Regarding the calling sequence, it should be noted that the ID must be placed in bit positions 18-23 of its half word and that the 'no more vle' return occupies two half words in the calling sequence (for an optional linkage to a fix-up subroutine) instead of the usual one half word for the non-normal return.

## DETAILS

The location of the first bit of the vle with the same ID as that in the calling sequence will be in the value field of the index word, Anextx.


RSERCH
GET ADDRESS OF SYMBOL
1
HOUSEKEEPING
READ CALLING SEQUENCE
2
DETERMINE FIRST ENTRY LOCATION
3
UPDATE BY ENTRY LENGTH
4
EQUAL TO GIVEN LOCATION
NO
5
LAST ENTRY
YES
6
INCREASE $X15 BY 1.32
YES
7
PUT ADDRESSES IN CONTROL BLOCK
8
EXIT
9
INDEX SPECIFIED
NO
10
INCREASE $X15 BY 2.0
YES
11
MOVE DICTIONARY ENTRY
12
UPDATE INDEX REGISTER

FLOW CHART DESCRIPTION

Box 1

$0 - $7 are saved and bit Azzs is set to one to indicate to TABMAN that the management routine being used is RSERCH. The contents of the index register I in the calling sequence is saved for later tests.

Boxes 2-6

The location of the first dictionary location is determined and its function address is compared to the address given in the calling sequence. If it is not equal and if we have not exhausted the list, we update by the entry length, and test the next dictionary entry. If there are no more entries in the list, we set up $15 for the not found return, and branch to the exit routine.

Boxes 7, 9-12

If the function address specified in the calling sequence has been found we place the dictionary location and the function location in the appropriate words in the table control block. Also, if an index register was specified in the calling sequence, the dictionary entry is moved to the location specified in the value field of the index register, and the value field is updated by the length of the entry moved. In either case $15 is set up for the found return, and a branch is made to the exit routine.

Box 8

$0 - $7 are restored. Return to the user program is a -- B,-1($15).

Boxes 2-4

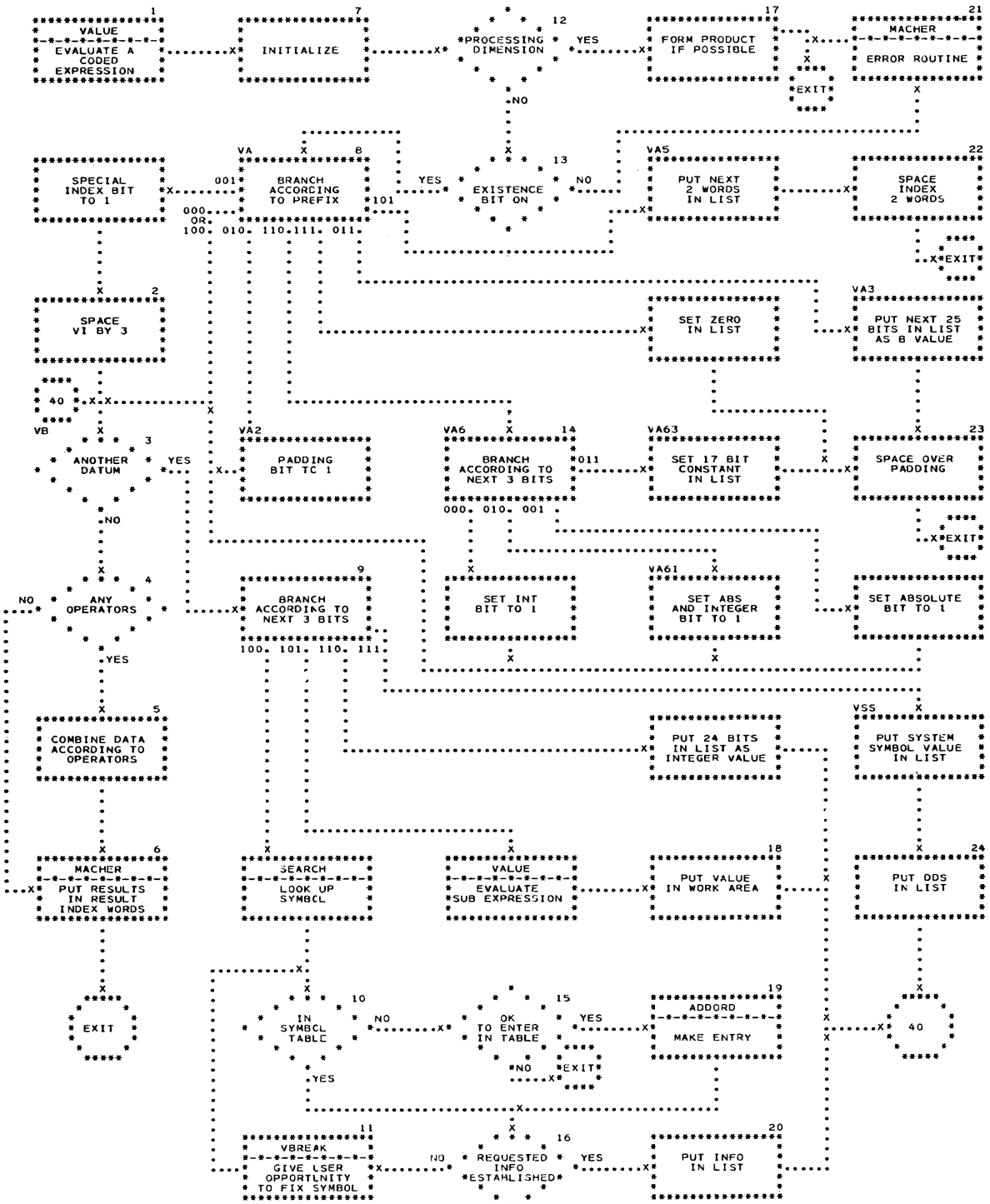
If bit Afirst is on, $2 is reset to the first location of the vle table. If off the location of the next vle entry is picked up from index word Anexta. The bit is turned off by the test.

Boxes 5, 6

If the ID in the calling sequence is not equal to the ID in the entry and more entries are in the table, $2 is updated by the length of the entry just tested and the next entry ID is checked. If there are no more entries the routine exits to the "no more entries" return.

Boxes 8, 9

If the ID's are equal and the entry is not deleted, the location of the entry is put in the index word Anextx and the index word Anexta is set to point to the next vle. Exit is made to the normal exit. If the entry has been deleted we treat it as if it had not been found.


1
VALUE
EVALUATE A CODED EXPRESSION
7
INITIALIZE
12
PROCESSING DIMENSION
YES
NO
17
FORM PRODUCT IF POSSIBLE
EXIT
21
MACHER
ERROR ROUTINE
VA
8
SPECIAL INDEX BIT TO 1
001
000 OR 100. 010. 110. 111. 011.
BRANCH ACCORDING TO PREFIX
101
YES
13
EXISTENCE BIT ON
NO
VA5
PUT NEXT 2 WORDS IN LIST
22
SPACE INDEX 2 WORDS
EXIT
2
SPACE VI BY 3
SET ZERO IN LIST
VA3
PUT NEXT 25 BITS IN LIST AS B VALUE
40
VB
3
ANOTHER DATUM
YES
NO
VA2
PADDING BIT TC 1
VA6
14
BRANCH ACCORDING TO NEXT 3 BITS
011
000. 010. 001
VA63
SET 17 BIT CONSTANT IN LIST
23
SPACE OVER PADDING
EXIT
4
ANY OPERATORS
NO
YES
9
BRANCH ACCORDING TO NEXT 3 BITS
100. 101. 110. 111.
SET INT BIT TO 1
VA61
SET ABS AND INTEGER BIT TO 1
SET ABSOLUTE BIT TO 1
5
COMBINE DATA ACCORDING TO OPERATORS
PUT 24 BITS IN LIST AS INTEGER VALUE
VSS
PUT SYSTEM SYMBOL VALUE IN LIST
6
MACHER
PUT RESULTS IN RESULT INDEX WORDS
SEARCH
LOOK UP SYMBCL
VALUE
EVALUATE SUB EXPRESSION
18
PUT VALUE IN WORK AREA
24
PUT DDS IN LIST
EXIT
10
IN SYMBCL TABLE
NO
YES
15
OK TO ENTER IN TABLE
YES
NO
EXIT
19
ADDORD
MAKE ENTRY
40
11
VBREAK
GIVE LSER OPPORTUNITY TO FIX SYMBOL
NO
16
REQUESTED INFO ESTABLISHED
YES
20
PUT INFO IN LIST

PURPOSE

VALUE evaluates a coded expression
VSKIP skips over a coded expression.
VDIMEN evaluates a dimension statement.
VMOVE moves a coded expression.

LINKAGE

VSKIP

On Entry

1. The calling sequence is:
LVI, 15, $+2; B, VSKIP
2. The index word Valinx locates the first bit of the coded expression.

On Exit

1. Valinx will be updated to locate the first bit after the coded expression.
2. If the using program has turned on bit, Vcundf, VSKIP also will have made a symbol table entry for any undefined symbol encountered in the coded expression.

VDIMEN

On Entry

1. The calling sequence is:

| | |
|---|---|
| LVI, 15, $+2; B, VDIMEN | |
| B, a fix-up subroutine | @not evaluated exit |
| Normal Return | @not evaluated exit |

2. $1 locates the first bit of the dimension statement.

On Exit

1. If the dimension statement was evaluated, $1 contains the location of the product of the dimensions.
2. If the dimension statement could not be evaluated, $1 still locates the first bit of the dimension statement.
3. If the using program has turned on bit Vcundf, VDIMEN also will have made a symbol table entry for any undefined symbol encountered in the coded expression.

VMOVE

On Entry
1. The calling sequence is:
LVI, 15, $+2; B, VMOVE
Normal Return
2. The index word, Valinx, locates the first bit of the coded expression.
3. The index word, Vmovex, contains the location where the expression is to be moved.

On Exit
1. After the expression has been moved, Valinx and Vmovex will be updated accordingly.
2. If the using program has turned on bit, Vcundf, VMOVE also will have made a symbol table entry for any undefined symbol encountered in the coded expression.

VALUE

On Entry

1. The calling sequence is:
LVI, 15, $+2; B, VALUE
B, a fix-up subroutine @break return
Normal Return
2. The index word Valinx locates the first bit of the coded expression.
3. Bit Vcbv = 1 if VALUE is not to attempt to get value and index information.
Bit Vcbsx = 1 if VALUE is not to attempt to get S/X type information.
Bit Vcbdim = 1 if VALUE is not to attempt to get dimension information.
Bit Vcbdds = 1 if VALUE is not to attempt to get dds information.

On Exit
1. If while attempting to do its job, VALUE discovers that a symbol table entry is lacking needed information, VALUE will exit to the break exit and provide the using program with the following information:
a. $1 will locate the function part of the symbol table entry causing the trouble.
b. Bit Vnbv = 1 if value and index information is needed.
Bit Vnbsx = 1 if S/X type information is needed.
Bit Vnbdim = 1 if dimension information is needed.
Bit Vnbdds = 1 if dds information is needed.

c. Bit Vfbv = 1 if the value and index information is affected.
Bit Vfbsx = 1 if S/X type of information is affected.
Bit Vfbdim = 1 if dimension information is affected.
Bit Vfbdds = 1 if dds information is affected.

d. Bits Vcbc, Vcbsx, Vcbdim, Vcbdds, are not changed.

2. When VALUE exits to the normal return, VALUE provides the using program with the following information:

| | Location | dds | Contents |
|---|---|---|---|
| value and index information | Sini | (B, 25) | integer value |
| | Sinb | (B, 25) | bit style value |
| | Sinx | (BU, 4) | index |
| | Sinxq | (BU, 1) | = 1 if there is an index<br>= 0 if there is no index |
| | Sinrl1 | (BU, 1) | relocation indicators* |
| | Sinrl2 | (BU, 1) | |
| S/X type information | Sinsx | (BU, 2) | 00 integer style (S)<br>10 bit style (X)<br>01<br>11 |
| dimension information | Sindim | (B, 25) | dimension reference |
| dds information | Sindds | (B, 25) | dds reference |
| | Sindsq | (BU, 1) | = 1 if the dds reference is known<br>= 0 if it is not known |
| | Sinser | (BU, 4) | symbol error flags |

* See Section on relocation ops.

3. If VALUE had been unable to find some of the desired information, the bit Vfbnot will be on to indicate this fact and the appropriate Vfb bits will be set to show what information could not be found.

4. If the using program had turned on bit Vcunfd, VALUE also will have made a symbol table entry for any undefined symbol encountered in the coded expression.

5. Valinx will be updated to locate the first bit after the coded expression.

DETAILS

Break exit: Following a break exit, the using program may take any action it desires, including (perhaps):

1. changing the setting of Vcbv, Vcbsx, Vcbdim and Vcbdds to show that certain information is no longer desired, or

2. filling in missing information into the symbol table.

If desired, VALUE may be used at this time to evaluate other coded expressions. Eventually, the using program must return to one of the locations, Vpunt or Vrtrn, after the break exit from VALUE. By a Vrtrn return if the situation has improved (i.e., if any of the Vcb bits have been turned on, or if further information has been added to the symbol table or both), then VALUE tries again where it previously ran into trouble. If the situation has not been improved, VALUE notes what items can no longer be evaluated, and then proceeds with the evaluation of the others,

if any. By using the Vpunt return to VALUE after a break exit, the user program effectively has all the Vcb bits turned on and the Vrtrn procedure performed.

Treatment of Undefined Symbols: The treatment is dependent on the setting of the control bit Vcundf.

1. If Vcundf is on when an undefined symbol is encountered, a standard symbol table entry will be made, and then VALUE will proceed as though it had found this entry there. Note: Even if all the Vcb bits are on, VALUE will still look at all symbols and make this entry when needed.
2. If Vcundf is off when an undefined symbol is encountered, VALUE will note what items can no longer be computed, and will proceed without making a break.

Example: The coded expression is for A+B. The using program wants all the information available from VALUE about the expression.

Discovering that no information is in the symbol table for A, VALUE makes a break exit with all the Vfb and Vnb bits on. Then the program fills in values into the symbol table, and goes to Vrtrn. Whereupon, VALUE discovers that the situation is improved, but still all the desired information is not yet there. Again VALUE makes a break exit, now with the bits Vfbsx, Vfbdim, Vfbdds, Vnbdim, and Vnbdds on. The user program decides to ignore the dimension reference and returns to Vrtrn with bit Vcbdim on. Once again VALUE sees that the situation is improved, but still some desired information is lacking. Therefore VALUE makes a break exit with the bits Vfbsx, Vfbdds, Vnbsx, and Vnbdds on. On this occasion the user program returns directly back to Vrtrn. Seeing that there is no improvement VALUE makes note that the S/X type and dds type information is not available and continues through the coded expression of the B datum. VALUE makes no further attempt to look up S/X type information. However, if B has a dimension reference, the fact that A was ambiguous on this point is no longer relevant. Note: When a break is taken, it is not meaningful for the user program to turn off any of the Vcb bits. If the user program does this, his action will be ignored by VALUE.

| Item | Type | Purpose |
|---|---|---|
| Valinx | XW | Locates coded expression. |
| Vmovex | XW | Locates the position to which the coded expression is to be moved. |
| Vcbv<br>Vcbsx<br>Vcbdim<br>Vcbdds | bits | Tell VALUE what information not to attempt to get. |
| Vnbv<br>Vnbsx<br>Vnbdim<br>Vnbdds | bits | Tell the using program what information could not be found in the symbol table (when a break exit is made). |
| Vfbv<br>Vfbsx<br>Vfbdim<br>Vfbdds | bits | Tell the using program what is not computable in the coded expression unless information stated by the Vnb bits is supplied (when a break exit is made). |
| Vfbnot | bit | Is on if anything was not found that should have been found (when the normal exit is made). |
| Vcundf | bit | Is on if VALUE is to enter undefined symbols in the symbol table. |

FLOW CHART DESCRIPTION

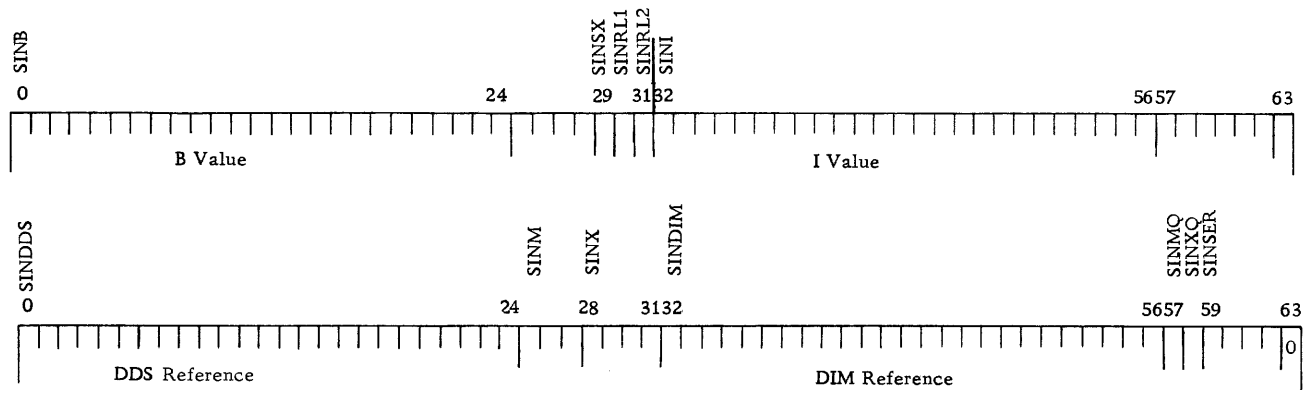
VALUE has a 'push' type list with values being 'pushed' upwards as they are computed or combined. The first five words of the list are called the heading; the workspace follows. After the first workspace there is a sequence of headings and workspaces determined by the number of existence bits encountered by VALUE in the coded expression. $10 points to the location of the current heading; $11 points to the location in the workspace.

Box 12

Branch-on-zero-bit Valdfg (indicating special entry made by VDIMEN) to box 13. The VDIMEN subroutine deals with the dimension reference statement for a user outside of VALUE.

Box 17

Get address of dimension reference, set control bits, and branch to VALUE to evaluate the coded expression. If VALUE evaluates the expression, form product of dimensions, and turn on completed bit; if not, go to VBREAK.




| | | | | | |
|---|---|---|---|---|---|
| SINDDS | (BU, 25) | DDS Reference | SINSER | (BU, 4) | Symbol Error Flags |
| SINM | (BU, 3) | Mode Bits | SINB | (BU, 25) | B Value |
| SINX | (BU, 4) | Index | SINSX | (BU, 1) | Subscript/Index |
| SINDIM | (BU, 25) | Dimension Reference | SINRL1 | (BU, 1) | Relocation Flag 1 |
| SINMQ | (BU, 1) | Mode Bits Given | SINRL2 | (BU, 1) | Relocation Flag 2 |
| SINXQ | (BU, 1) | Index Given | SINI | (BU, 25) | I Value |

Figure 9. Format of VALUE's Output (without Relocation Fields)

Box 13

Branch to MACHER subroutine if existence bit is off. (Existence bit is the first bit in the coded expression and must be on.)

Box 8

Pick up the coded expression prefix (bits two, three, and four in the coded expression), space the pointer over the prefix, and branch according to the prefix.

| Coded Expression Prefix | Type of Coded Expression |
|---|---|
| 000 | Normal case |
| 001 | Special case |
| 010 | Padded case |
| 011 | Special signed |
| 100 | Super special |
| 101 | Fully evaluated |
| 110--- | Long prefix |
| 111 | Null |

Box 14

If the coded expression prefix is 110, then we must branch according to the next three bits (bits five, six, and seven in the expression) as we had a long prefix.

| Additional Bits in a Long Prefix | Type of Coded Expression |
|---|---|
| 000 | Integer |
| 001 | Absolute |
| 010 | Absolute integer |
| 011 | Seventeen bit constant |

Box 3

Test for another datum by seeing if the no-more-datum bit was set off by GETFLD. If off, go to box 4. If there is another datum, turn off Vvsymb (1 - if last datum is a symbol) and the dimension reference flags: Vvcdr (1 - if current datum is a symbol whose dimension was asked for and obtained), Vvcdrl (1 - if current datum is a symbol whose dimension was asked for and not obtained), and Vvcdrm (1 - if current datum is a symbol whose dimension was not asked for). Then branch to box 9.

Boxes 4, 5

If either Vcbv or Vtbv is on, we do not want to pute the value of the total coded expression. So we see if Vcbsx is on (indicating we do not want subscript/index type). If neither the value nor the S/X is desired, go to box 6. If either Vcbv or Vcbsx is on, set up correct table address for arithmetic (or S/X) routine. If there is another operator, pick up

the operator code from the coded expression and get both the section entry and the operand type from the table. Then after picking up the datum number from the coded expression and getting the location of the right and/or left operand if needed, perform the indicated operation. When there is no other operator, go to box 6.

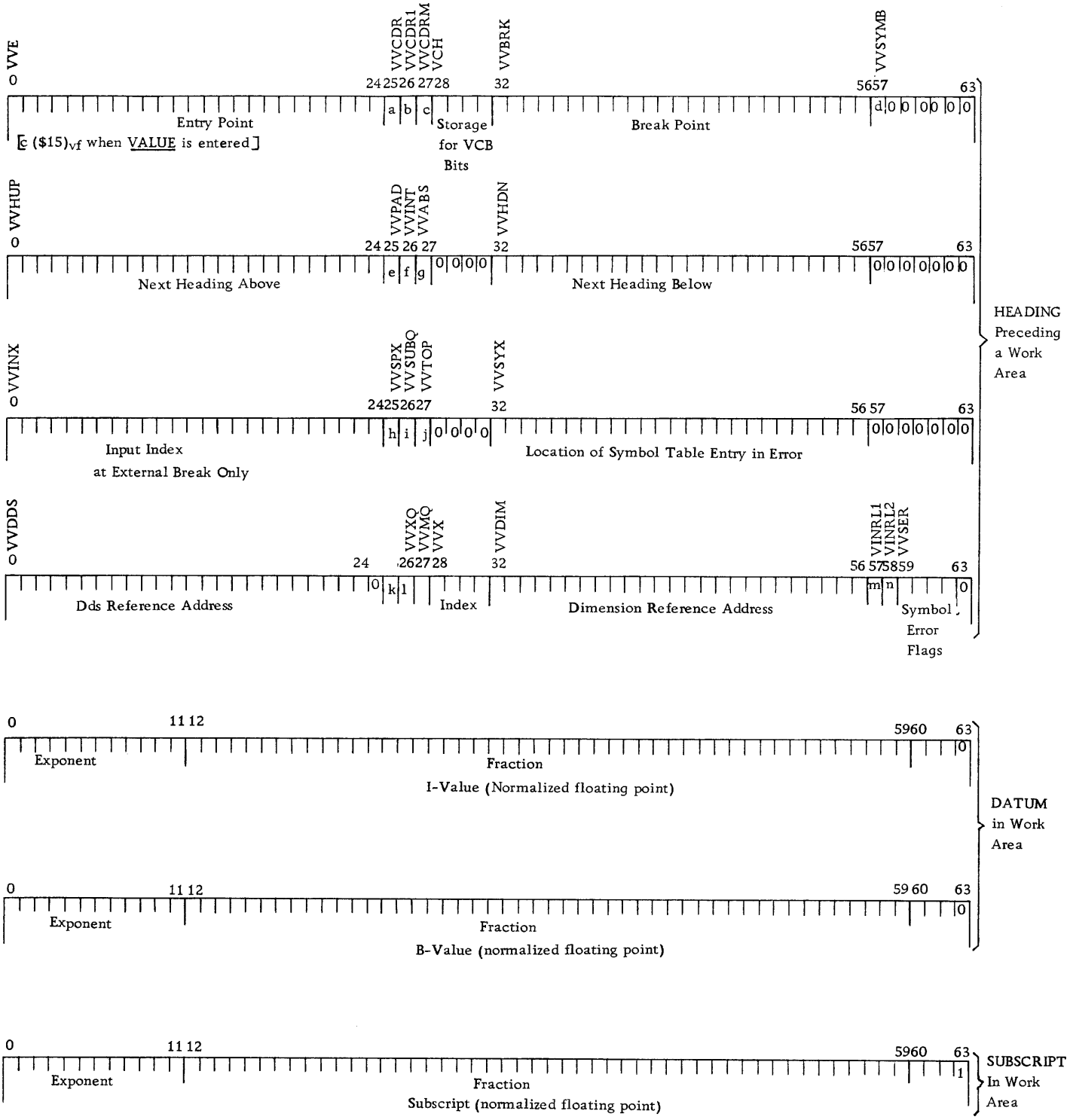
Box 6

Reset $11 (current location in workspace) with $10 (location of current heading). If S/X type is unknown, set on trouble bit, Vtbsx. If no values were computed, go to Vz. Mark index as existing if Vvspx is on. If Vvspx is off and Vvabs is on (indicating result is to be made), set signs of the integer and of the bit values to plus, and continue. If the result is not to be made an integer, i.e., Vvint is off, then exit; otherwise combine integer and bit values, and set bit value to zero. Skip the pointer over the padding in the coded expression. Put integer and bit values into VALUE's output words, Sini and Sinb.

Box 9

Pick up datum prefix (usually bits five, six, and seven in the coded expression), space the pointer over the prefix and branch according to the prefix.

| Datum Prefix | Type of Datum |
|---|---|
| 100 | Symbol |
| 101 | Subexpression |
| 110 | Absolute |
| 111-- | Long |
| 11100 | System symbol |



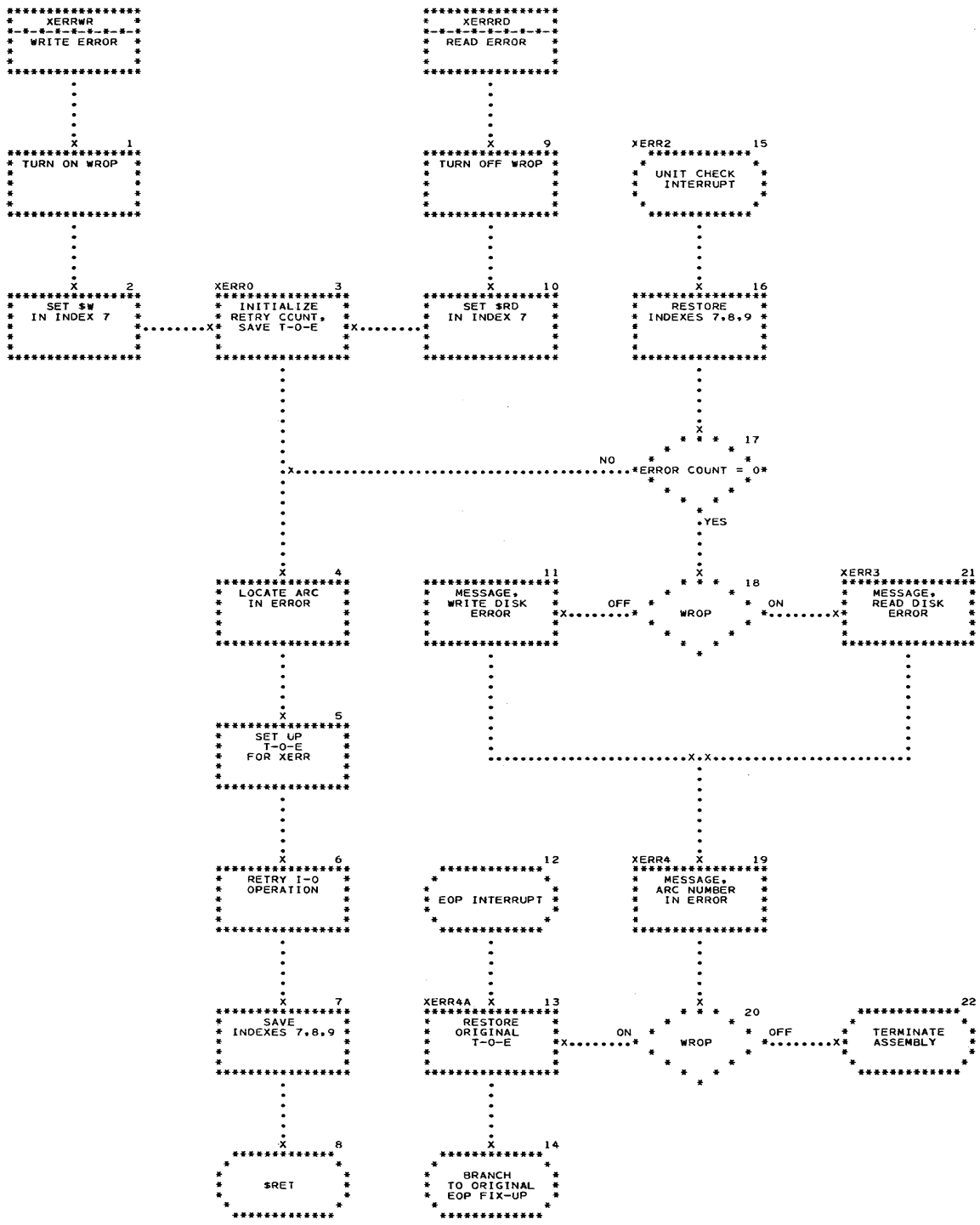

Legend

a. dimension requested and obtained
b. dimension requested and not obtained
c. dimension not requested
d. last datum symbol
e. padding
f. result integer
g. result absolute
h. result index
i. datum with subscripts
j. entry from outside
k. index
l. mode known
m. relocation
n. relocation

ABLE:

See section on relocation ops for format of fifth word in heading.

Figure 10. Format of VALUE's Internal Tables


XERRWR
WRITE ERROR
XERRRD
READ ERROR
1
TURN ON WROP
9
TURN OFF WROP
XERR2 15
UNIT CHECK INTERRUPT
2
SET $W IN INDEX 7
XERR0 3
INITIALIZE RETRY CCUNT, SAVE T-O-E
10
SET $RD IN INDEX 7
16
RESTORE INDEXES 7,8,9
17
ERROR COUNT = 0
NO
YES
4
LOCATE ARC IN ERROR
11
MESSAGE, WRITE DISK ERROR
18
WROP
OFF
ON
XERR3 21
MESSAGE, READ DISK ERROR
5
SET UP T-O-E FOR XERR
6
RETRY I-O OPERATION
12
EOP INTERRUPT
XERR4 19
MESSAGE, ARC NUMBER IN ERROR
7
SAVE INDEXES 7,8,9
XERR4A 13
RESTORE ORIGINAL T-O-E
20
WROP
ON
OFF
22
TERMINATE ASSEMBLY
8
$RET
14
BRANCH TO ORIGINAL EOP FIX-UP

LINKAGE

The user routine, when receiving a unit check, must set up in the UK fixup the error arc number in the value field of $8, and the address of the I-O control word in the value field of $9. If the error occurred during a read operation, the unit check fixup must branch to Xerrrd, or to Xerrwr if it occurred during a write operation.

DETAILS

The operation will be retried a maximum of 100 times. If not corrected in this time, the assembly will be terminated (in the case of read errors) or continued with error (in the case of write errors).

FLOW CHART DESCRIPTION

Boxes 1, 9

The Wrop indicator is set according to the operation to be retried. If on, a write operation is indicated; if off, a read.

Boxes 2, 3, 10

The I-O operation is loaded into the value field of $7. A count of 100 is placed in the count field of $8, and the table of exits set up by the user routine is saved in location Xerr9.

Boxes 4-8

The arc in error is located; the EOP interrupt is suppressed. A new table of exits is set up specifying a branch to Xerr4a in the case of an EOP interrupt, and a branch to Xerrz in the case of unit check. The I-O operation is initiated, $7, $8, and $9 are saved in location Xerr5, and return to the interrupted address is made via $RET.

Box 12

This is location Xerr4a and is entered upon successful completion of the I-O operation initiated by XERR.

Boxes 13, 14

The original table of exits is restored as set up by the user routine, and XERR branches to the EOP fixup which would have been taken if the I-O error had not occurred.

Box 15

This is location Xerr2 and is entered upon XERR receiving a unit check from its I-O operation.

Box 16

$7, $8, and $9 are refilled with the I-O parameters saved in location Xerr5.

Box 17

The count in $8 is decremented by one. If the count is not zero, the I-O operation will be attempted again. If zero, the error has not been successful after 100 tries.

Boxes 18, 11, 21, 19

An appropriate message "Uncorrected Error Reading/ Writing Disk," and "Relative Arc Number XXXXXX" is written via System Output.

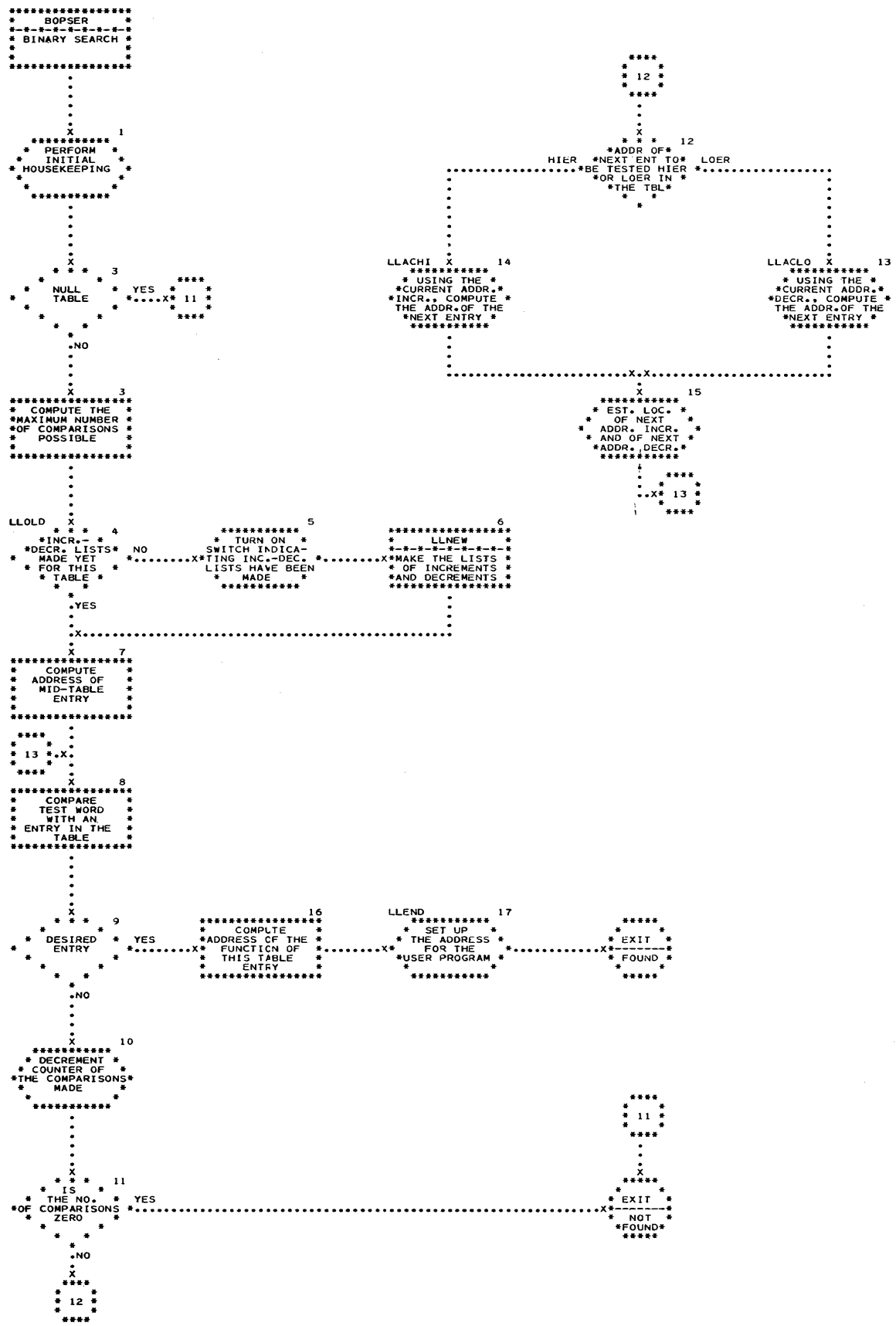
Boxes 20, 22

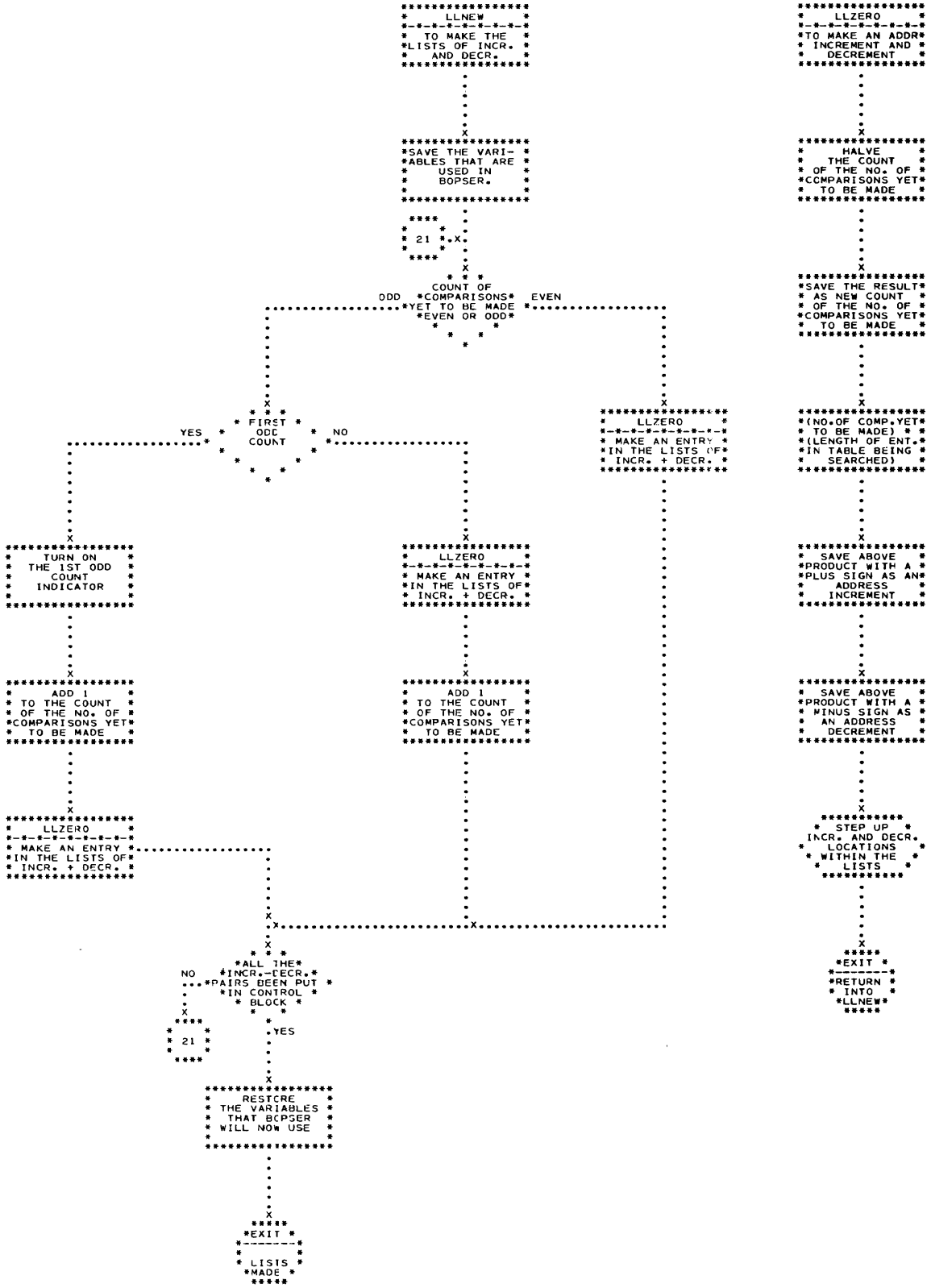
If a write operation, return will be made to the user routine's EOP fixup. If a read operation, the assembly is terminated.

MAJOR LOGIC AREAS IN THE PASS 1 PHASE OF STRAP II

| MAIN | MAIN flow for Pass 1 | Includes the Processing in the Following Subroutines and Subprocedures |
|---|---|---|
| BOPSER | Binary OPeration SEaRch | Performs the binary search on the operation tables and on the system symbol table. |
| CCA8 | conversion of CC to A8 code | Converts code of card input to the A8 internal processing code of STRAP II. |
| CPLTNM | ComPLeTe a NuMber | Fetches the remaining characters of the number from the card image input and computes, using the radix specified, the number represented. |
| CPLTSY | ComPLeTe a SYmbol | Fetches and collects in a specified buffer the remaining characters of a programmer's symbol with its tail or gets the value and data description of a system symbol. |
| GETCHA | GET a CHAracter | Gets a character from the input card image for main flow to process. |
| GETFLD | GET a FieLD encoded | Does the Pass 1 encoding put in the expanded instruction units. |
| GNLOAD | Get Name LOADed | Creates the name file and writes it on disk. |
| INTOUT | OUTput the INTermediate expanded instruction unit | Buffers the expanded instruction units and writes them on disk for later processing during Pass 2. |
| MBSPEC | Main's SPECial character substitution | Scans an illegal op or an illegal system symbol for a possible error character and substitutes in its place the corresponding option character. |
| MIBA | Main's Indicator Branch instruction Analysis | Performs the branch-on indicator analysis on any operation which MAIN does not find in the operation table. |
| MIOD | Main's IOD processing | Processes the Master Control Program (MCP) pseudo-ops during Pass 1. |
| MQDD | Main's DD procedure | Processes the DD during Pass 1; takes care of the radix, entry mode, multiple-numeric cases, the special sign (S), and the special (X) in particular. |
| MQDALF | Main's ALPHabetic Data definition procedure | Converts the alphabetic data definition statement from the internal A8 code to the code specified in the entry mode; puts results in the expanded instruction unit. |
| MQDNUM | Main's processing for a NUMeric Data definition | Evaluates the statement field of a numeric DD to the terminal character, to the special sign or the special exponent. |
| DNUM | complete a Data definition's NUMeric field | Collects the characters of an operand from the D field of a numeric DD and transforms the specified operand into a double precision floating point number. |
| MQDDPA | Main's DD Precision Arithmetic subroutines | Performs the precision arithmetic specified between the fields of a numeric DD. |
| MQP | Main's Punsym procedure | Collects in the unit the symbols specified by the PUNSYM. |
| MQR | Main's data Reservation procedure | Processes the address field of a DR or a DRZ, that is, sets up the dimension information. |
| MDIMRF | Main's DIMension ReFerence procedure | Makes a primitive dimension reference entry for a dimension specified by a DR or DRZ without a name. |
| MDIMRT | | Makes a primitive dimension reference entry for a dimension specified by a SYN or by a DR or DRZ with a name. |
| MQS | Main's Syn procedure | |

| | | |
|---|---|---|
| MULTI | MULTIply-defined symbols | Adjusts the symbol table after MAIN has discovered that a symbol is multiply-defined. |
| MX | Main's eXtract procedure | Has the expressions formed for the extract parameters in their own unit; then has the statement of the EXTract pseudo-op set up in a second unit. |
| TAILOR | TAIL OR untail procedure | Updates the information determining the special tailing configuration appended by MAIN or by CPLTSY during Pass 1 to each symbol in the sphere of influence of a TAIL pseudo-op. |
| XINIT | INITialization for input/output subroutines | Initializes the Input-Output routines for Pass 1. |
| XINPUT | INPUT another card image | Brings into memory each new card image; has each new card image converted into the internal A8 processing code. |


BOPSER
BINARY SEARCH
1
PERFORM INITIAL HOUSEKEEPING
2
NULL TABLE
YES
11
NO
3
COMPUTE THE MAXIMUM NUMBER OF COMPARISONS POSSIBLE
LLOLD
4
INCR.- DECR. LISTS MADE YET FOR THIS TABLE
NO
5
TURN ON SWITCH INDICATING INC.-DEC. LISTS HAVE BEEN MADE
6
LLNEW
MAKE THE LISTS OF INCREMENTS AND DECREMENTS
YES
7
COMPUTE ADDRESS OF MID-TABLE ENTRY
13
8
COMPARE TEST WORD WITH AN ENTRY IN THE TABLE
9
DESIRED ENTRY
YES
16
COMPLTE ADDRESS OF THE FUNCTION OF THIS TABLE ENTRY
LLEND
17
SET UP THE ADDRESS FOR THE USER PROGRAM
EXIT FOUND
NO
10
DECREMENT COUNTER OF THE COMPARISONS MADE
11
IS THE NO. OF COMPARISONS ZERO
YES
EXIT NOT FOUND
12
ADDR OF NEXT ENT TO BE TESTED HIER OR LOER IN THE TBL
HIER
LOER
LLACHI
14
USING THE CURRENT ADDR. INCR., COMPUTE THE ADDR. OF THE NEXT ENTRY
LLACLO
13
USING THE CURRENT ADDR. DECR., COMPUTE THE ADDR. OF THE NEXT ENTRY
15
EST. LOC. OF NEXT ADDR. INCR. AND OF NEXT ADDR. DECR.


LLNEW
TO MAKE THE LISTS OF INCR. AND DECR.
SAVE THE VARIABLES THAT ARE USED IN BOPSER.
21
COUNT OF COMPARISONS YET TO BE MADE EVEN OR ODD
ODD
EVEN
FIRST ODD COUNT
YES
NO
TURN ON THE 1ST ODD COUNT INDICATOR
ADD 1 TO THE COUNT OF THE NO. OF COMPARISONS YET TO BE MADE
LLZERO
MAKE AN ENTRY IN THE LISTS OF INCR. + DECR.
LLZERO
MAKE AN ENTRY IN THE LISTS OF INCR. + DECR.
ALL THE INCR.-DECR. PAIRS BEEN PUT IN CONTROL BLOCK
NO
21
YES
RESTORE THE VARIABLES THAT BOPSER WILL NOW USE
EXIT
LISIS MADE
LLZERO
TO MAKE AN ADDR INCREMENT AND DECREMENT
HALVE THE COUNT OF THE NO. OF COMPARISONS YET TO BE MADE
SAVE THE RESULT AS NEW COUNT OF THE NO. OF COMPARISONS YET TO BE MADE
LLZERO
(NO.OF COMP.YET TO BE MADE) (LENGTH OF ENT. IN TABLE BEING SEARCHED)
SAVE ABOVE PRODUCT WITH A PLUS SIGN AS AN ADDRESS INCREMENT
SAVE ABOVE PRODUCT WITH A MINUS SIGN AS AN ADDRESS DECREMENT
STEP UP INCR. AND DECR. LOCATIONS WITHIN THE LISTS
EXIT
RETURN INTO LLNEW

## LINKAGE

### On Entry

The calling sequence is:

```
SIC,$15;B,BOPSER
,address of the
   control block (I)
B, a fix-up subroutine      'not found exit
                              from BOPSER.
Normal Return               'found exit from
                              BOPSER.
```

A control block must be set up in the format shown in Figure 11.

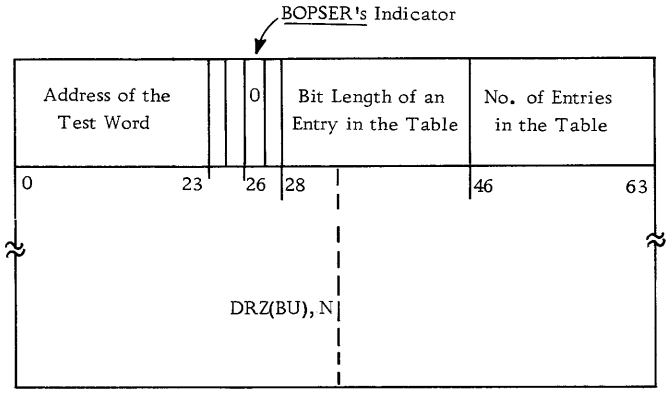



Figure 11. Format of a Control Block for BOPSER

### On Exit

BOPSER puts the address of the function of the found entry in the value field, Llxw.

## RESTRICTIONS

1. The table to be searched must be ordered.
2. $N = \lceil \log_2 (\text{no. of entries in the table to be searched}) \rceil$.
3. The test word must be 64 bits in length.
4. The minimum length of an entry in the table to be searched is 64 bits.

## USERS

MAIN: to search the primary op table and the secondary op table.

CPLTSY: to search the system symbol table.

## FLOW CHART DESCRIPTION

### Box 1

The housekeeping includes setting up three index registers, $7, $8, and $9, with the address of the list of address increments, with the address of the table to be searched, and with the address of the list of address decrements, respectively. The address of the test word is put in the compare instruction.

### Boxes 2, 3

The test for a null table also establishes a left zero's count with which the count of the maximum number of searches through the table is calculated. (18-LZC = maximum number of tests.) This count is saved in the count field of $7.

### Boxes 4 - 6

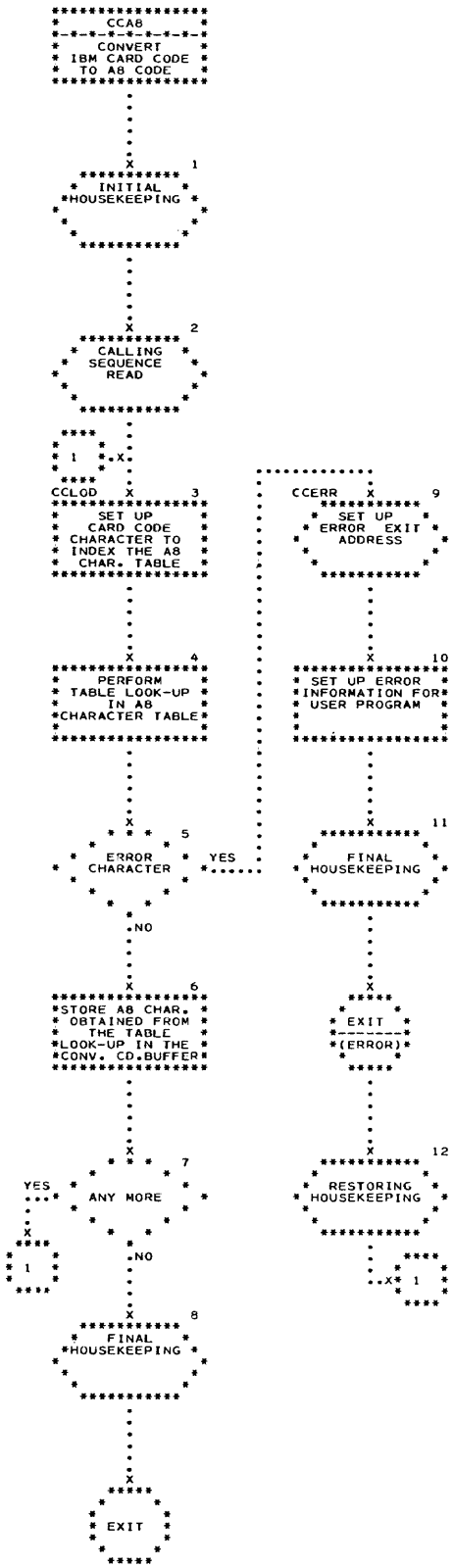
An indicator, turned on only by BOPSER, in the control block tells whether this table has been searched by BOPSER before in the assembly and thus whether or not the address increment-decrement pairs are already in the control block.

The first time BOPSER searches a new table, the subroutine makes its own set of address increment-decrement pairs which BOPSER saves in the space provided by the user program in his table control block.

Subsequently, whenever BOPSER searches the table specified by this control block, BOPSER uses these address increment-decrement pairs in directly computing the address of each entry to be inspected during the binary search through the table.

### Box 7

To the address of the past table test entry is applied the successive address increment and/or address decrement during the search. Because of the way the lists of binary search address modifiers are made, the location of the first entry to be tested is computed by adding the first address increment not to the actual starting location of the table but to an adjusted starting location. (The adjusted starting location is the location of the table minus the length of an entire entry.)


CCA8
CONVERT IBM CARD CODE TO A8 CODE
1
INITIAL HOUSEKEEPING
2
CALLING SEQUENCE READ
1
CCLOD
3
SET UP CARD CODE CHARACTER TO INDEX THE A8 CHAR. TABLE
4
PERFORM TABLE LOOK-UP IN A8 CHARACTER TABLE
5
ERROR CHARACTER
YES
NO
6
STORE A8 CHAR. OBTAINED FROM THE TABLE LOOK-UP IN THE CONV. CD. BUFFER
7
ANY MORE
YES
NO
1
8
FINAL HOUSEKEEPING
EXIT
CCERR
9
SET UP ERROR EXIT ADDRESS
10
SET UP ERROR INFORMATION FOR USER PROGRAM
11
FINAL HOUSEKEEPING
EXIT (ERROR)
12
RESTORING HOUSEKEEPING
1

## LINKAGE

Initially

The calling sequence is:

```
SIC, $15; B, CCA8
, A(I)                  'address of IBM card code
                          character buffer.
, B(I)                  'address of A8 character
                          buffer.
, n(I)                  'n = an integer (without radix
                          point) specifying count of
                          IBM card code characters
                          to be converted.
B, a fixup subroutine   'error return.
Normal Return
```

Re-entry

1. The contents of $15 must be preserved outside of CCA8 because the contents of $15 are used in the re-entry linkage.
2. The legal 12-bit character to be substituted for the illegal IBM card code must be loaded into the accumulator at offset 107 before re-entry.
3. After re-entering the CCA8 subroutine with a B,0($15) instruction, CCA8 will resume the converting process with the substitute IBM card code character.

## USER

XINPUT: to convert the card image of each card type input into a card image in A8 code, the processing code of STRAP II.

## FLOW CHART DESCRIPTION

Boxes 1, 8, 11, 12

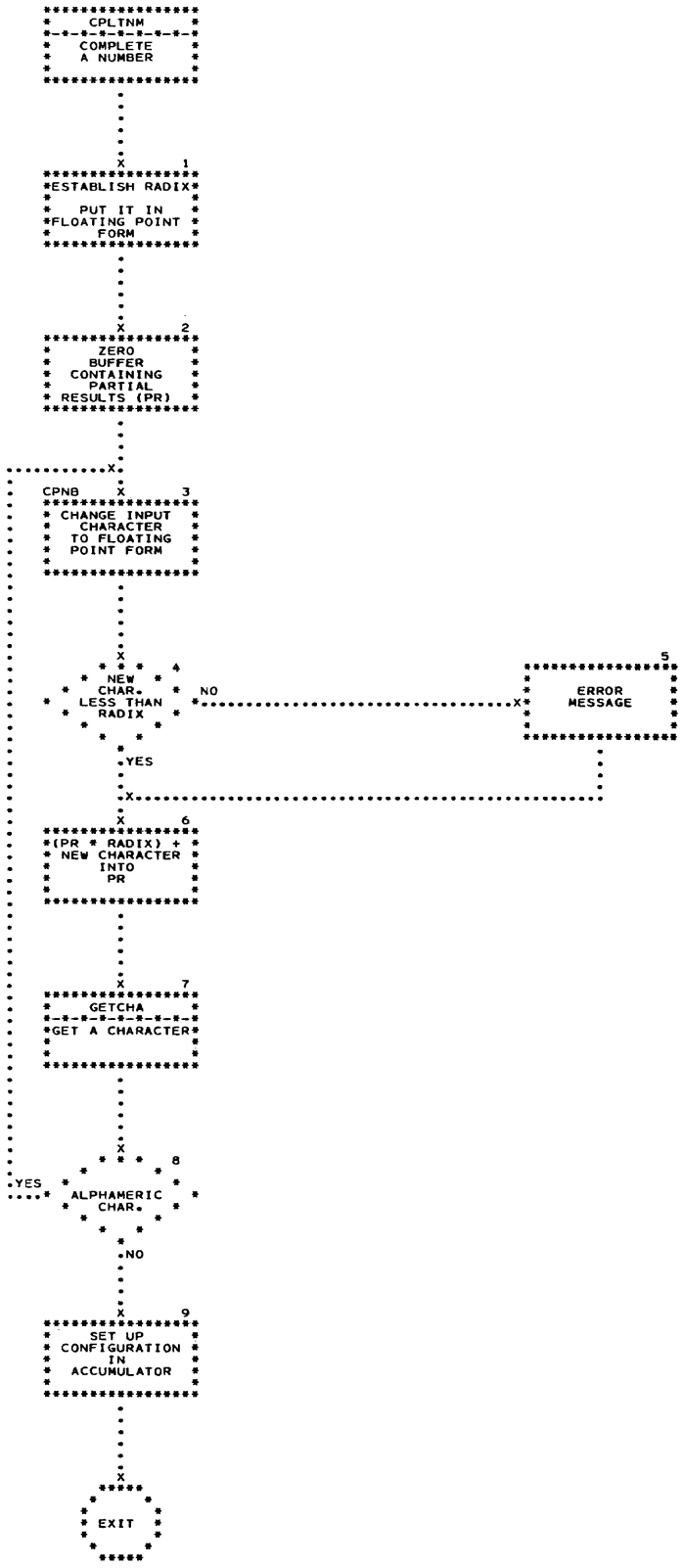
Indicator bits, mask bits, and index registers are preserved and restored by CCA8.

Box 5

The table of A8 characters has a field of eight zeros for each position in the table that can be referenced by an illegal IBM card code character. (Note that if this generalized subroutine -- and it definitely is one -- should be used out of STRAP II, the subroutine could be easily adjusted to do a slower conversion with a much shorter table of A8 characters by inserting instructions before those for the table look-up to test for illegal multiple punches in the high-order three positions of each IBM card code character.)

Boxes 9-11

If an illegal IBM card code character is detected by CCA8 during the table look-up, a reference index word is set up in the right half of the accumulator. The value field contains the address of the next IBM card code character; the count field contains the number of characters yet to be converted (excluding the illegal one). CCA8 then exits back into the user routine.


CPLTNM
COMPLETE A NUMBER
1
ESTABLISH RADIX
PUT IT IN FLOATING POINT FORM
2
ZERO BUFFER CONTAINING PARTIAL RESULTS (PR)
CPNB
3
CHANGE INPUT CHARACTER TO FLOATING POINT FORM
4
NEW CHAR. LESS THAN RADIX
NO
5
ERROR MESSAGE
YES
6
(PR * RADIX) + NEW CHARACTER INTO PR
7
GETCHA
GET A CHARACTER
8
ALPHAMERIC CHAR.
YES
NO
9
SET UP CONFIGURATION IN ACCUMULATOR
EXIT

LINKAGE

On Entry

1. The calling sequence is:
   SIC, Cpltnz; B, CPLTNM.
2. The first character of the number is in the value field of $3.
3. The radix is in the 8-bit field, Gfsrad.

On Exit

1. The first character following the number is in the value field of $3.
2. The number itself is left as a 24-bit number in the accumulator at offset 64, with an additional prefix of 3 marker bits signifying an absolute datum.

USERS

| | |
|---|---|
| GETFLD: | to get a number specified in a field. (GETFLD does not collect the numeric characters in the statement field of a numeric DD.) |
| MAIN: | to get the number with a leading F-entry mode. |
| MQDD: | to get the number in a field radix of a DD. |
| MQDD: | to get the number in a field g.p. of a DD. |

FLOW CHART DESCRIPTION

Box 1

Since the intermediate arithmetic calculations are in floating point, the radix specified in Gfsrad is converted to a floating point number and saved. CPLTNM accepts a radix as high as 16 where the letters A-F represent the values 10-15. If the Gfsrad field is zero, CPLTNM uses a radix of 10. The Gfsrad field is one of the areas that is zeroed in the initialization prior to the STRAP II processing of an instruction.

Box 2

The partial result area where the intermediate results are built up is zeroed.

Boxes 3 - 8

The loop is performed for each 'digit' character that belongs to the numeric datum. Each input character for the number is changed into floating point form for the floating point calculation.

The input character (now in floating point form) is checked to see if it is less than the radix (now in floating point form) specified. If the digit equals or exceeds the radix, an error message is executed, but the digit is still used in the computation of the numeric datum.

The necessary arithmetic computation is performed and the results become the new contents of the previous results operand.

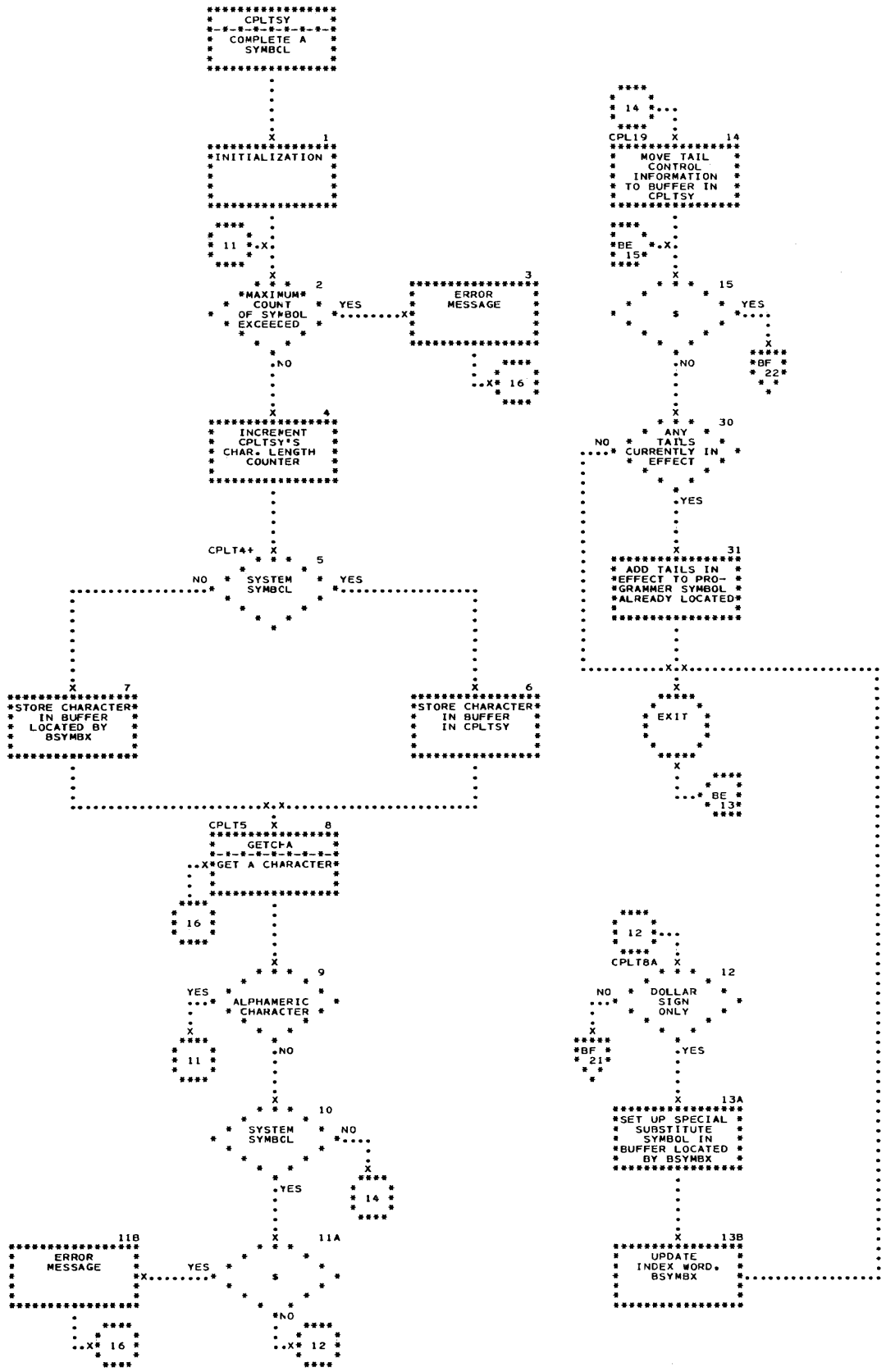
CPLTNM fetches each of its characters from the input by GETCHA which gives its user the characters in the value field of $3. When CPLTNM receives a non-alphameric character from GETCHA, CPLTNM considers the characters of the numeric datum completely collected.

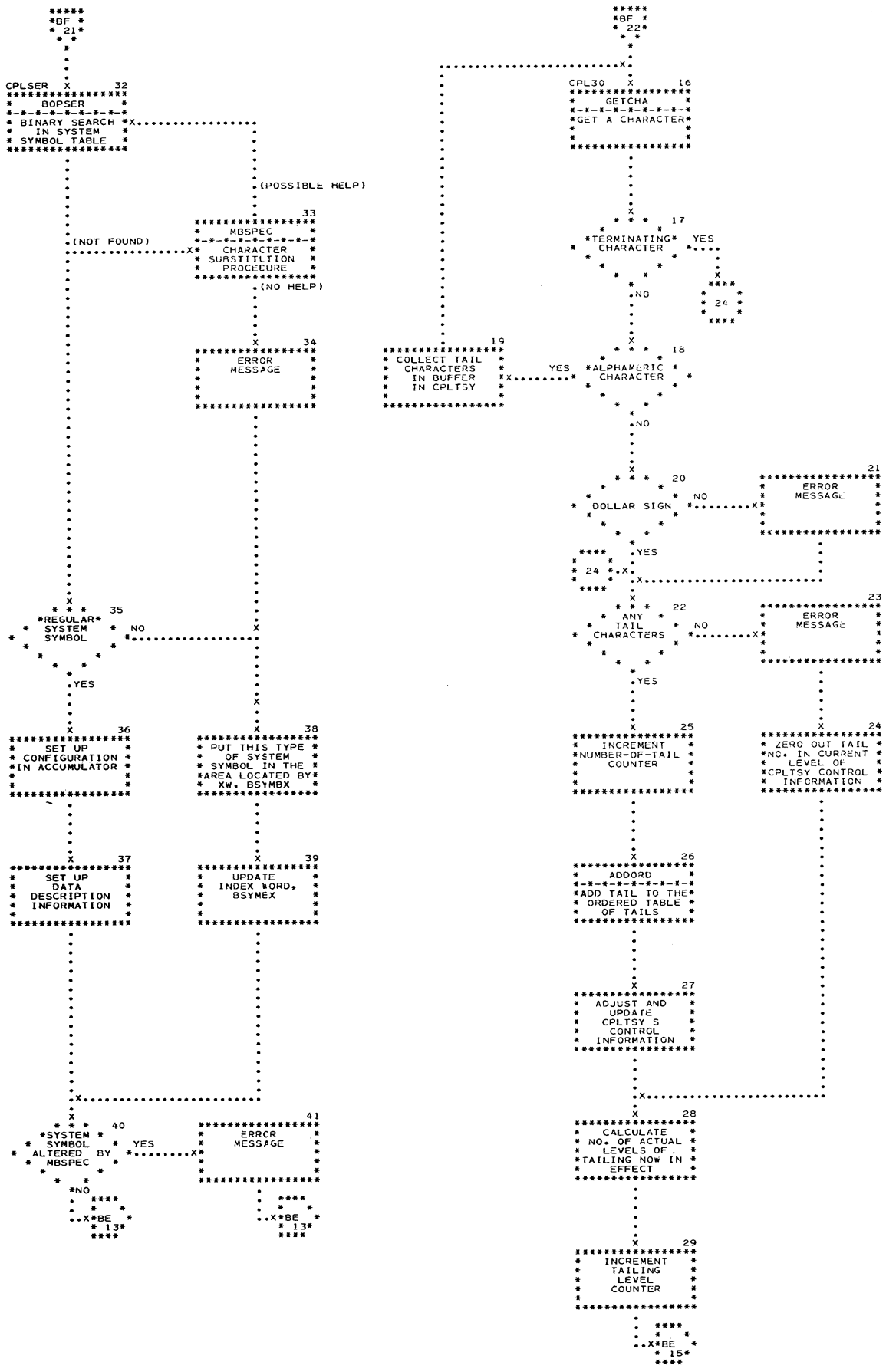
Box 9

Adjusting the final partial result, CPLTNM sets up a specific configuration in the accumulator at offset 64 in order that the using program can, if necessary, directly insert the numeric information into a coded expression as an absolute datum of the coded expression.

The final configuration at offset 64 in the accumulator is shown in Figure 12.

Figure 12. Format of CPLTNM's Output in $L


CPLTSY
COMPLETE A SYMBOL
1
INITIALIZATION
11
2
MAXIMUM COUNT OF SYMBOL EXCEEDED
YES
3
ERROR MESSAGE
16
NO
4
INCREMENT CPLTSY'S CHAR. LENGTH COUNTER
CPLT4
5
SYSTEM SYMBOL
NO
YES
7
STORE CHARACTER IN BUFFER LOCATED BY BSYMBX
6
STORE CHARACTER IN BUFFER IN CPLTSY
CPLT5
8
GETCHA
GET A CHARACTER
16
9
ALPHAMERIC CHARACTER
YES
11
NO
10
SYSTEM SYMBOL
NO
YES
14
11B
ERROR MESSAGE
YES
11A
$
NO
16
12
14
CPL19
MOVE TAIL CONTROL INFORMATION TO BUFFER IN CPLTSY
BE 15
15
$
YES
NO
BF 22
30
ANY TAILS CURRENTLY IN EFFECT
NO
YES
31
ADD TAILS IN EFFECT TO PRO- GRAMMER SYMBOL ALREADY LOCATED
EXIT
BE 13
12
CPLT8A
DOLLAR SIGN ONLY
NO
YES
BF 21
13A
SET UP SPECIAL SUBSTITUTE SYMBOL IN BUFFER LOCATED BY BSYMBX
13B
UPDATE INDEX WORD, BSYMBX


*BF * *21*
CPLSER X 32
BOPSER
BINARY SEARCH IN SYSTEM SYMBOL TABLE
(NOT FOUND)
(POSSIBLE HELP)
33
MBSPEC CHARACTER SUBSTITLTION PROCECURE
(NO HELP)
34
ERROR MESSAGE
REGULAR* SYSTEM SYMBOL
NO
YES
35
36
SET UP CONFIGURATION *IN ACCUMULATOR
37
SET UP DATA DESCRIPTION INFORMATION
38
PUT THIS TYPE OF SYSTEM SYMBOL IN THE *AREA LOCATED BY* XW, BSYMBX
39
UPDATE INDEX WORD, BSYMEX
40
SYSTEM SYMBOL ALTERED BY MBSPEC
YES
NO
41
ERRCR MESSAGE
*BE * 13*
*BF * *22*
CPL30 X 16
GETCHA
*GET A CHARACTER*
17
TERMINATING* CHARACTER
YES
NO
*BE * 24 *
19
COLLECT TAIL CHARACTERS IN BUFFER IN CPLTSY
YES
18
ALPHAMERIC CHARACTER
NO
20
DOLLAR SIGN
NO
YES
21
ERROR MESSAGE
22
ANY TAIL CHARACTERS
NO
YES
23
ERROR MESSAGE
24
ZERO OUT TAIL *NO. IN CURRENT LEVEL OF *CPLTSY CONTROL INFORMATION
25
INCREMENT *NUMBER-OF-TAIL COUNTER
26
ADDORD
*ADD TAIL TO THE* ORDERED TABLE OF TAILS
27
ADJUST AND UPDATE CPLTSY S CONTROL INFORMATION
28
CALCULATE NO. OF ACTUAL LEVELS OF *TAILING NOW IN EFFECT
29
INCREMENT TAILING LEVEL COUNTER
*BE * 15*

LINKAGE

On Entry

1. The calling sequence is:
   SIC, Cpltsz; B, CPLTSY.
2. The first character of the symbol, i.e., a $ or an alphabetic is in the value field of $3.
3. The location where the symbol is to be placed is in the value field of Bsymbx.

On Exit

See details under boxes 32-37.

USERS

| | |
|---|---|
| MAIN: | to collect the characters of a symbol which is in a P-mode data description; |
| MQP: | to collect the characters of a symbol which is in the address field of a PUNSYM pseudo-op; |
| MQDD: | to get the value of a system symbol which is in the address field of a numeric DD pseudo-op; |
| GETFLD: | to collect the characters of a programmer symbol or get the value of a system symbol. |

FLOW CHART DESCRIPTION

Box 1

During the initialization CPLTSY does the following: zeros the length-of-symbol counter, zeros the number of levels counter, zeros the if-a-system-symbol switch, sets the initial setting of NOP/B-type switches, and sets up index registers with the location of the buffer located by Bsymbx and the location of CPLTSY's word buffer.

If the character in $3 is a ($) indicating a system symbol is to be collected, then the if-a-system-symbol switch is turned on, the collection buffer of the system symbol is blanked, and the maximum count of system symbol characters is set to eight (excluding ($)). Otherwise the symbol to be collected is a programmer symbol and the maximum character count is set to 128.

Boxes 2-9

As CPLTSY gets each of the characters from GETCHA, CPLTSY collects the alphameric characters of the symbol in one of two buffers depending on whether the symbol is a system symbol or a programmer symbol.

If the symbol is a programmer symbol, it is immediately collected in the area located by the index word, Bsymbx. (This area will usually be in the variable portion of the current expanded instruction unit where GETFLD is building up a coded expression.) CPLTSY maintains a character length counter so CPLTSY can determine later how much of the current tails in effect, if any, can be appended to the programmer symbol. On detection of a non-alphameric character, CPLTSY stops getting characters from the input for the programmer symbol.

If the symbol is a system symbol, it is collected in a buffer area of CPLTSY in preparation for the later look up of the symbol in the system symbol table. Afterwards, CPLTSY may or may not put the system symbol in the area located by the index word Bsymbx. On detection of a non-alphameric character, CPLTSY has the presently collected symbol looked up.

CPLTSY cannot use all this information directly from the TAILOR table because the information may have to be temporarily adjusted just for this programmer symbol if the programmer symbol being collected has any tails specified with the symbol. Any tail specified with a programmer symbol overrules any existing tail on the same level.

Box 15

This test determines whether there is any tail specified with the programmer symbol.

Boxes 16-29

The cycle through these blocks is performed for each tail specified after the programmer symbol.

(The first character of the tail is checked to be an alphabetic character.) The characters of the tail are collected. The number-of-tail counter is incremented. Then the tail with this new number is entered into the table of tails. However, if the tail itself is in the table already, the new tail with the new number is not put in the table of tails; the number-

of-tail counter is decremented, so the control data of TAILOR will always contain the highest tail number currently used in the table of tails.

The number associated with the tail just collected is stored also with the CPLTSY tailing control data in the field representing the level effected. The contents of the level-of-tailing indicator set is used to select the proper field in the string of fields representing the levels. If just a ($) were specified for the tail, the appropriate field in CPLTSY's control data will be zeroed to indicate that on this level there is no tailing for this symbol.

The number of levels of tailing now in effect for this symbol is recomputed after the level control data has been adjusted. Also, the level-of-tailing counter is incremented in anticipation of a new tail with the programmer symbol.

After all the level control data effecting this programmer symbol has been adjusted, then CPLTSY takes the necessary steps to append the special characters to the programmer symbol collected.

Box 30

CPLTSY inspects its tailing control data to see if any tails should be appended to this programmer symbol.

Box 31

The length of the actual programmer symbol determines whether none, some, or all of the tails in effect can be indicated on the programmer symbol by the appended special characters.

Thus, if there are tails in effect, the programmer symbol may be appended with n + 1 eight bit characters, a $-character to indicate a tail is present, and then n eight-bit characters. This 'n' is determined by the last calculated number of actual levels of tailing now in effect for this programmer symbol. Each of these n fields contains the number of the tail associated with the level concerned. In fact, these n consecutive fields are taken from CPLTSY's tailing control data.

Boxes 32-37

After a non-alphameric character has appeared in the collection of a system symbol, then CPLTSY has the symbol looked up in the table of system symbols by BOPSER. Note the collection buffer for the system symbol was blanked in the initialization to insure the proper background characters for a short system symbol.

An entry for a mathematical constant-type system symbol contains the location within STRAP of the double precision floating point constant, while an entry for just a system symbol contains the machine location assigned to the system symbol.

If the system symbol in its evaluated form is in the table, CPLTSY does the following: Using the dds information, also in each entry, CPLTSY inserts this data in the location, Cpldds, where the user program may have access to it. Furthermore, before exiting CPLTSY sets up in the accumulator at offset zero, the datum for the coded expression being built up by the user program.

If the system symbol is not in the system symbol table, CPLTSY tries to get the symbol adjusted by MBSPEC. If still the adjusted symbol is not in the table, then the system symbol in its original form is put in the area located by Bsymbx and Bsymbx is updated. The latter storing is also done if the corresponding symbol in the system table is marked as one unavailable at the installation. (Of course, CPLTSY inserts the collected programmer symbol in the area located by Bsymbx too.)

The fact that Bsymbx has been updated indicates to the routine using CPLTSY that a system symbol has been collected for which there is no value or that a programmer symbol has been collected.

Boxes 12, 13a, 13b

After CPLTSY collects a symbol that turns out to be a single $-character, then CPLTSY sets up a special configuration of bits in the area located by Bsymbx (Figure 14).

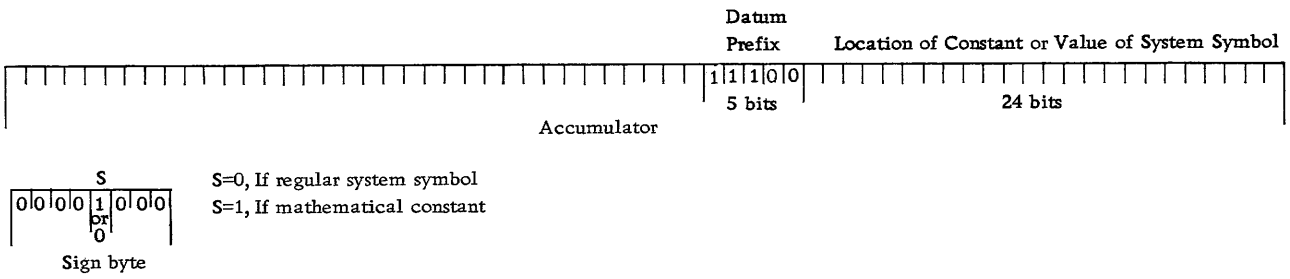



Figure 13. Format of CPLTNM's Output in $R

CPLTSY updates Bsymbx to locate the bit which CPLTSY has set to zero, immediately after the $00 configuration. Actually this configuration is to be part of the symbolic datum portion of a coded expression being built up by GETFLD. The GETFLD if-a-dollar-sign indicator, Gfdol, is turned on to distinguish the type of re-entry into GETFLD.

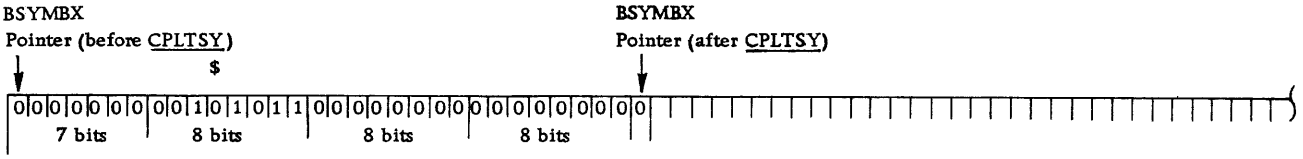
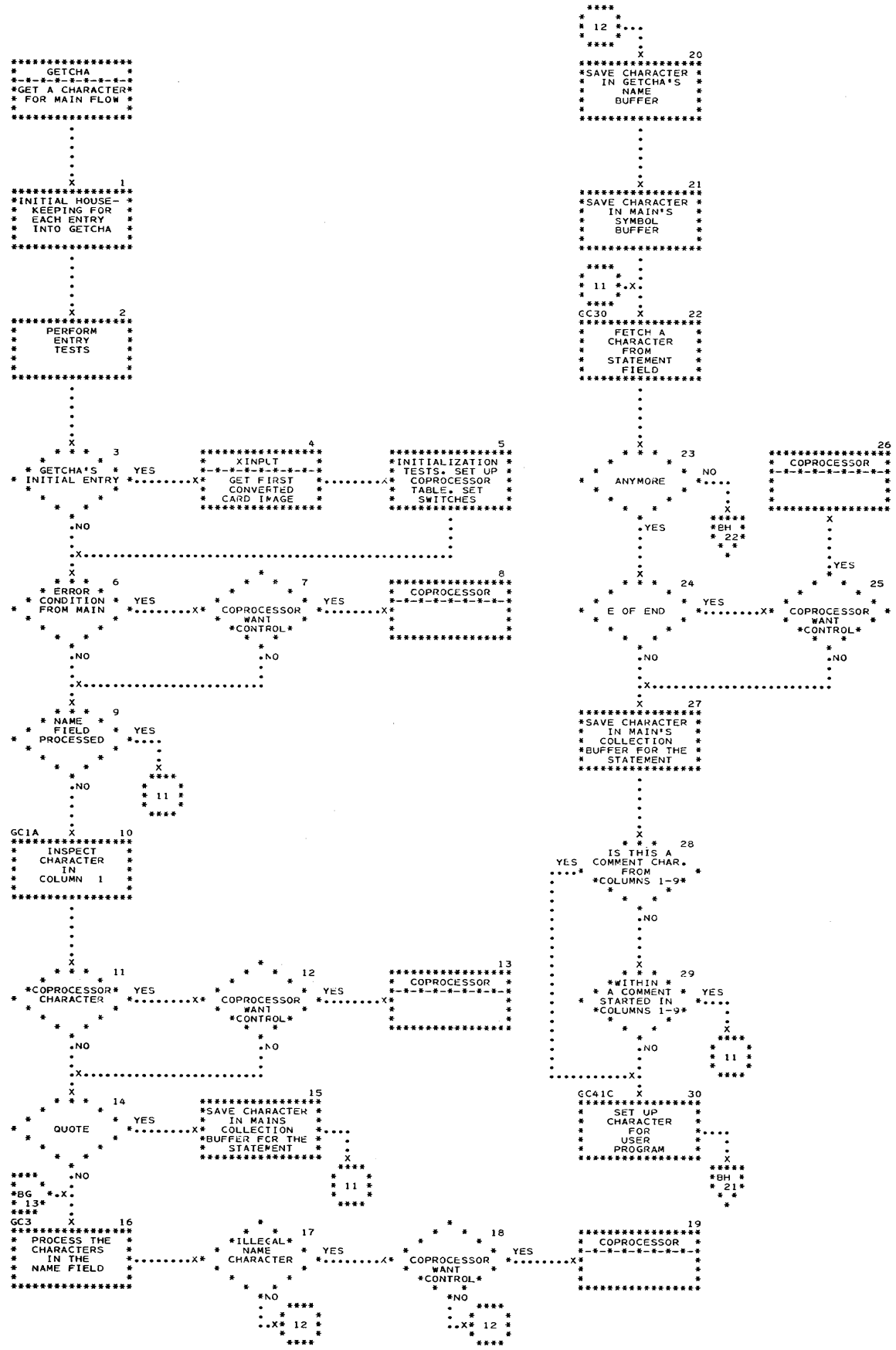
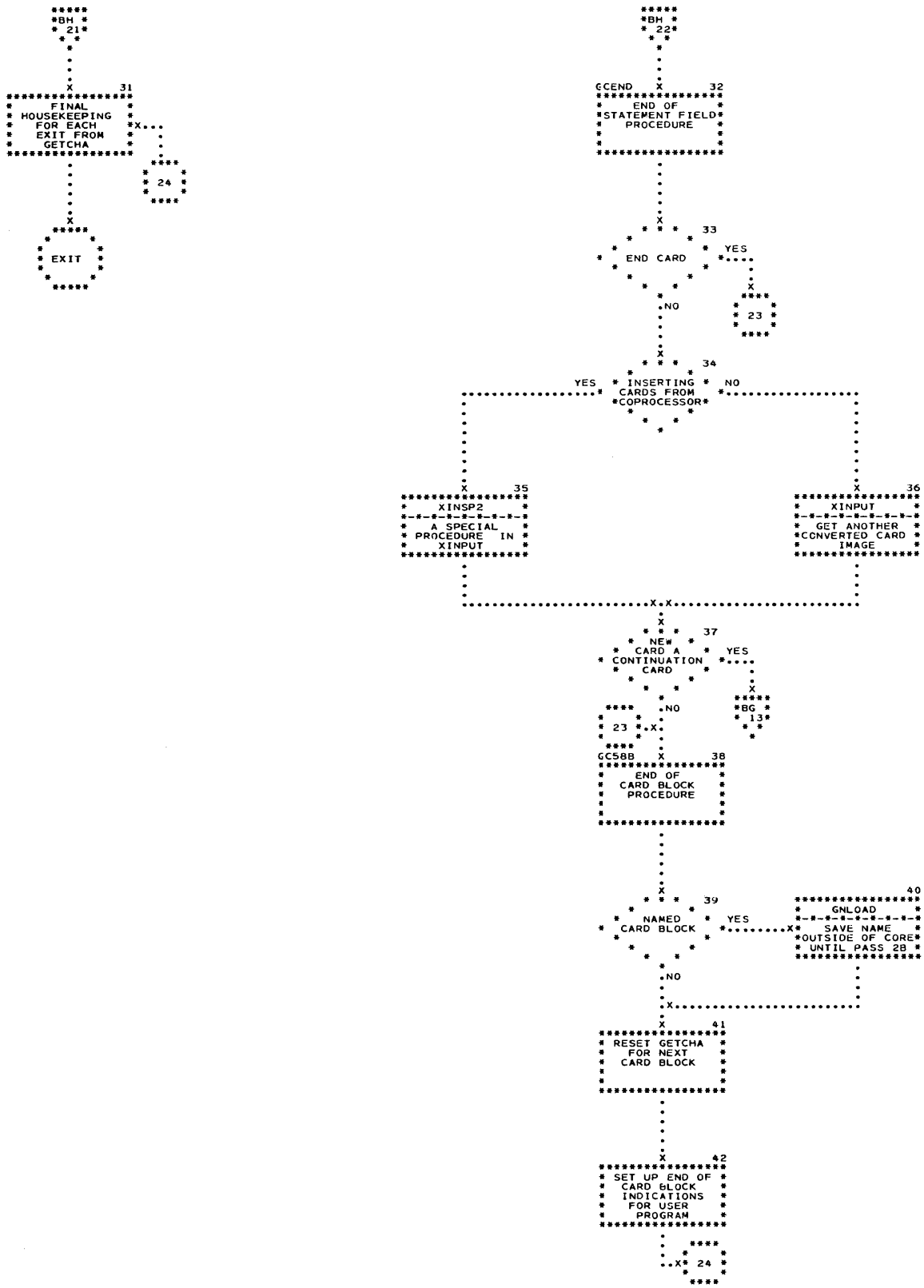



Figure 14. Format of CPLTSY's $ Output


GETCHA
GET A CHARACTER FOR MAIN FLOW
1
INITIAL HOUSE-KEEPING FOR EACH ENTRY INTO GETCHA
2
PERFORM ENTRY TESTS
3
GETCHA'S INITIAL ENTRY
YES
NO
4
XINPLT
GET FIRST CONVERTED CARD IMAGE
5
INITIALIZATION
TESTS. SET UP COPROCESSOR TABLE. SET SWITCHES
6
ERROR CONDITION FROM MAIN
YES
NO
7
COPROCESSOR WANT *CONTROL*
YES
NO
8
COPROCESSOR
9
NAME FIELD PROCESSED
YES
NO
10
GC1A
INSPECT CHARACTER IN COLUMN 1
11
COPROCESSOR CHARACTER
YES
NO
12
COPROCESSOR WANT *CONTROL*
YES
NO
13
COPROCESSOR
14
QUOTE
YES
NO
15
SAVE CHARACTER IN MAINS COLLECTION BUFFER FOR THE STATEMENT
11
*BG 13*
16
GC3
PROCESS THE CHARACTERS IN THE NAME FIELD
17
ILLEGAL NAME CHARACTER
YES
NO
18
COPROCESSOR WANT *CONTROL*
YES
NO
19
COPROCESSOR
20
SAVE CHARACTER IN GETCHA'S NAME BUFFER
21
SAVE CHARACTER IN MAIN'S SYMBOL BUFFER
11
GC30
22
FETCH A CHARACTER FROM STATEMENT FIELD
23
ANYMORE
NO
YES
*BH 22*
24
E OF END
YES
NO
25
COPROCESSOR WANT *CONTROL*
NO
26
COPROCESSOR
YES
27
SAVE CHARACTER IN MAIN'S COLLECTION BUFFER FOR THE STATEMENT
28
IS THIS A COMMENT CHAR. FROM *COLUMNS 1-9*
YES
NO
29
*WITHIN A COMMENT STARTED IN COLUMNS 1-9*
YES
NO
11
GC41C
30
SET UP CHARACTER FOR USER PROGRAM
*BH 21*


8H 21
31
FINAL HOUSEKEEPING FOR EACH EXIT FROM GETCHA
24
EXIT
8H 22
GCEND
32
END OF STATEMENT FIELD PROCEDURE
33
END CARD
YES
NO
23
34
INSERTING CARDS FROM COPROCESSOR
YES
NO
35
XINSP2
A SPECIAL PROCEDURE IN XINPUT
36
XINPUT
GET ANOTHER CONVERTED CARD IMAGE
37
NEW CARD A CONTINUATION CARD
YES
NO
8G 13
23
GC58B
38
END OF CARD BLOCK PROCEDURE
39
NAMED CARD BLOCK
YES
NO
40
GNLOAD
SAVE NAME OUTSIDE OF CORE UNTIL PASS 2B
41
RESET GETCHA FOR NEXT CARD BLOCK
42
SET UP END OF CARD BLOCK INDICATIONS FOR USER PROGRAM
24

LINKAGE

The calling sequence is: SIC, Getchz; B, GETCHA.

DETAILS

GETCHA performs the following functions:

1. This subroutine gives the characters of the input to main flow character-by-character through the value field of $3 during Pass 1 in A8 code.
2. This subroutine collects the characters of each name associated with a card block. Then the GNLOAD subroutine collects the names and puts them out on tape or on disk. Unlike the symbol file, the name file does not remain in memory during all the passes. The NAMEIN subroutine brings the name file into memory for the OUTPUT subroutine at the end of Pass 2 when the listing is produced.
3. This subroutine also builds up the corresponding symbol for each name. With each of these symbols, the TABMAN subroutines build up the symbol file. The symbol file remains in memory for the entire assembly.
4. This subroutine also collects the characters of the statement field in the buffer, Bsyst. The MAIN routine later puts the contents of this buffer into the unique expanded instruction unit associated with the instruction.
5. This subroutine tells main flow of Pass 1 when an end of card block has been reached by setting the bit, Fend, and by putting an A8 semicolon ("false semicolon") into the value field of $3.
6. This subroutine informs the coprocessor subroutine when one of its conditions has been detected by GETCHA or by MAIN or by XINPUT.

FLOW CHART DESCRIPTION

Boxes 14-16, 28, 29

A comment character appearing in columns 2-9 on the initial card of a card block causes the entire card block to be treated as if the comment character had appeared in column 1. In fact, GETCHA superimposes a comment character over the first column character of the initial card image. The comment character appearing in the name field of the card image is blanked out.

A comment character appearing in columns 2-9 of a continuation card causes the remaining characters on the card to be treated as a comment. The name may be continued on the following continuation cards of that card block. The characters from the statement field of these subsequent continuation cards are treated as part of the comment.

A comment character appearing in column 1 on the initial card of the card block causes the entire card to be treated as a comment.

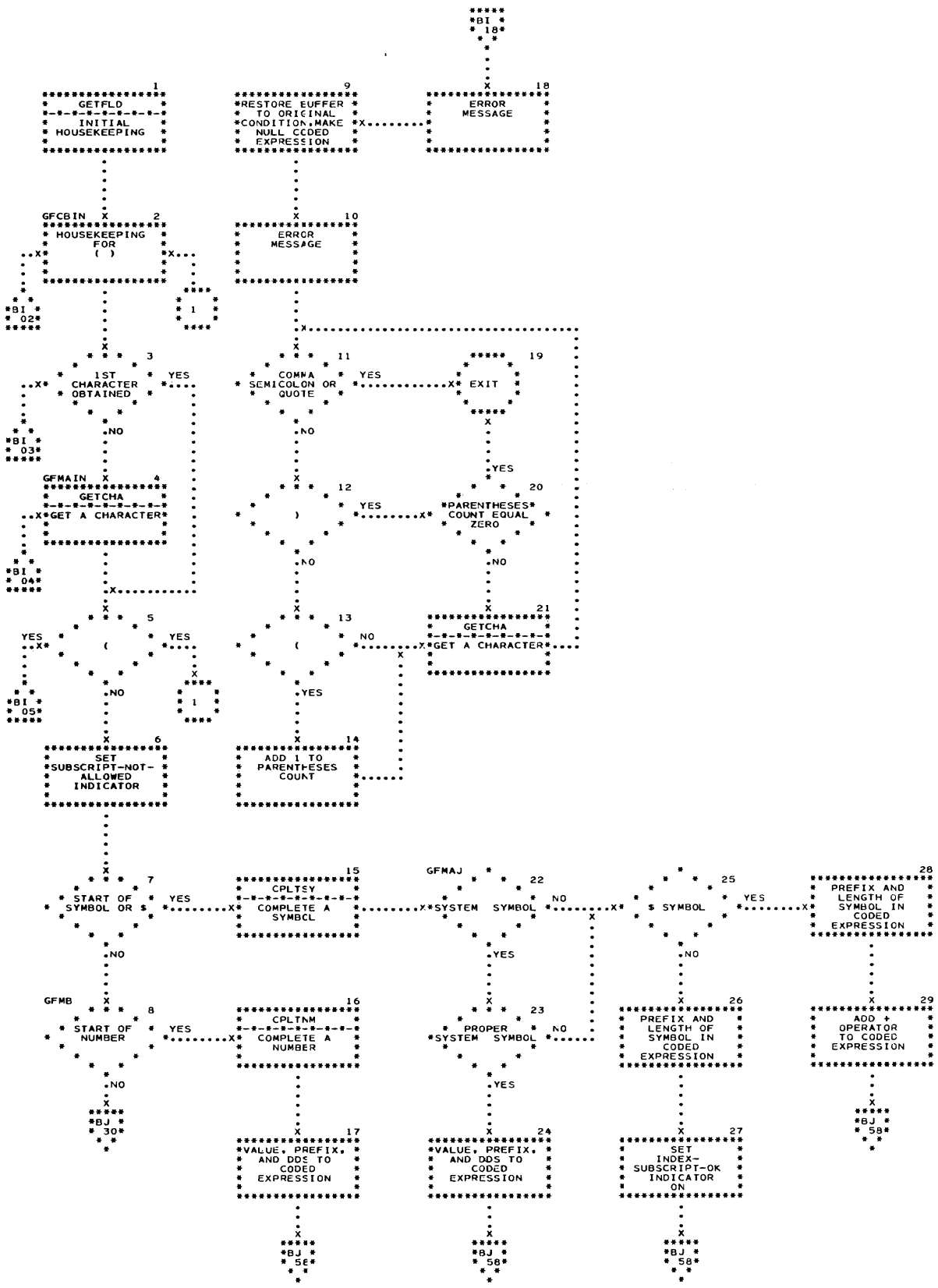
A comment character appearing in columns 10-72 causes the remaining character of the unit to be treated as part of the comment.

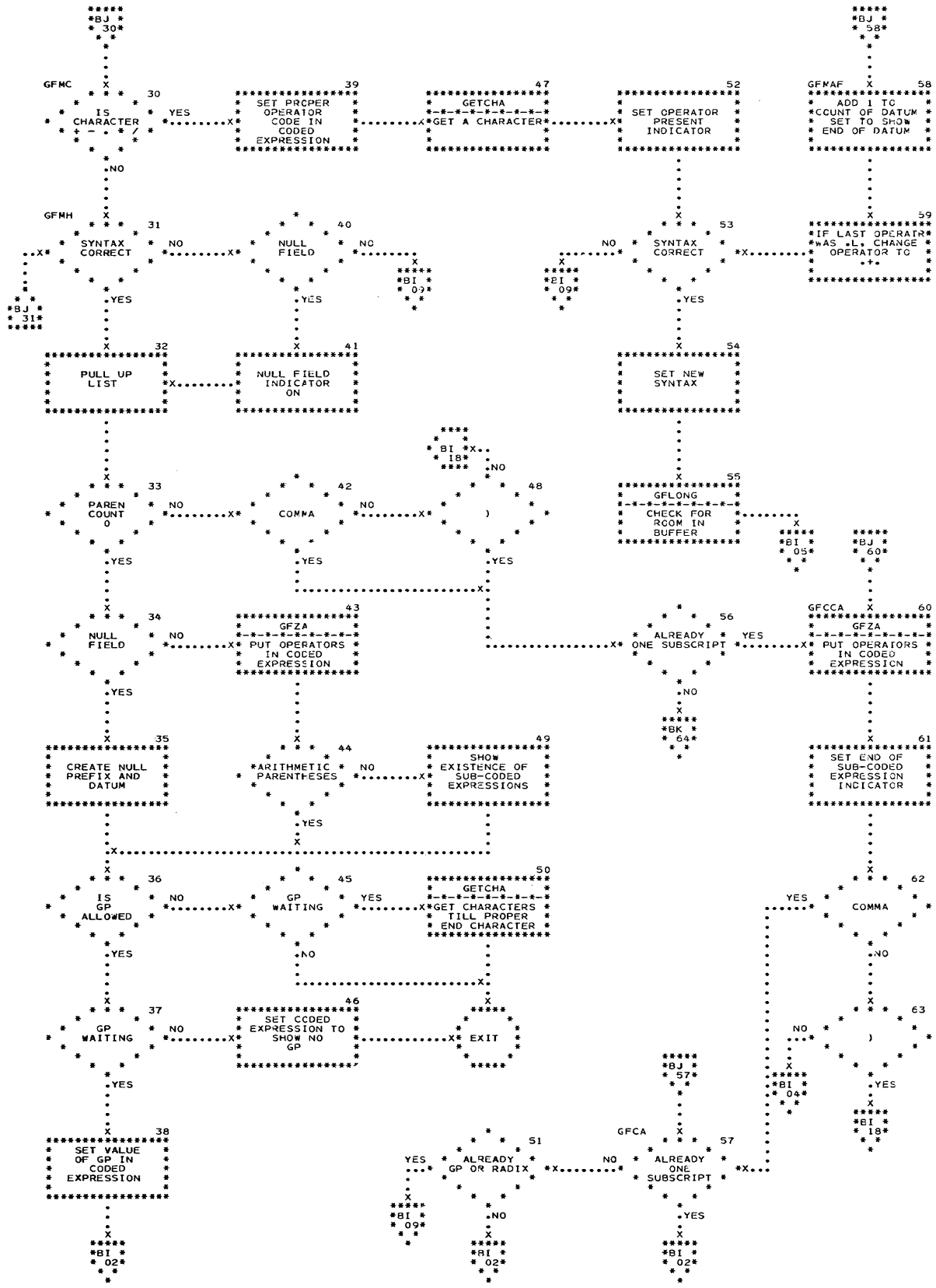
If a comment started in the first nine columns, then GETCHA informs main flow that a comment has been encountered with the comment character in $3 as usual. Subsequently, GETCHA will put all the remaining characters of this type of comment into the collection buffer for the statement characters without giving them out to main flow through $3. When GETCHA has finished collecting this type comment, GETCHA does inform main flow with the usual end of card block indications.

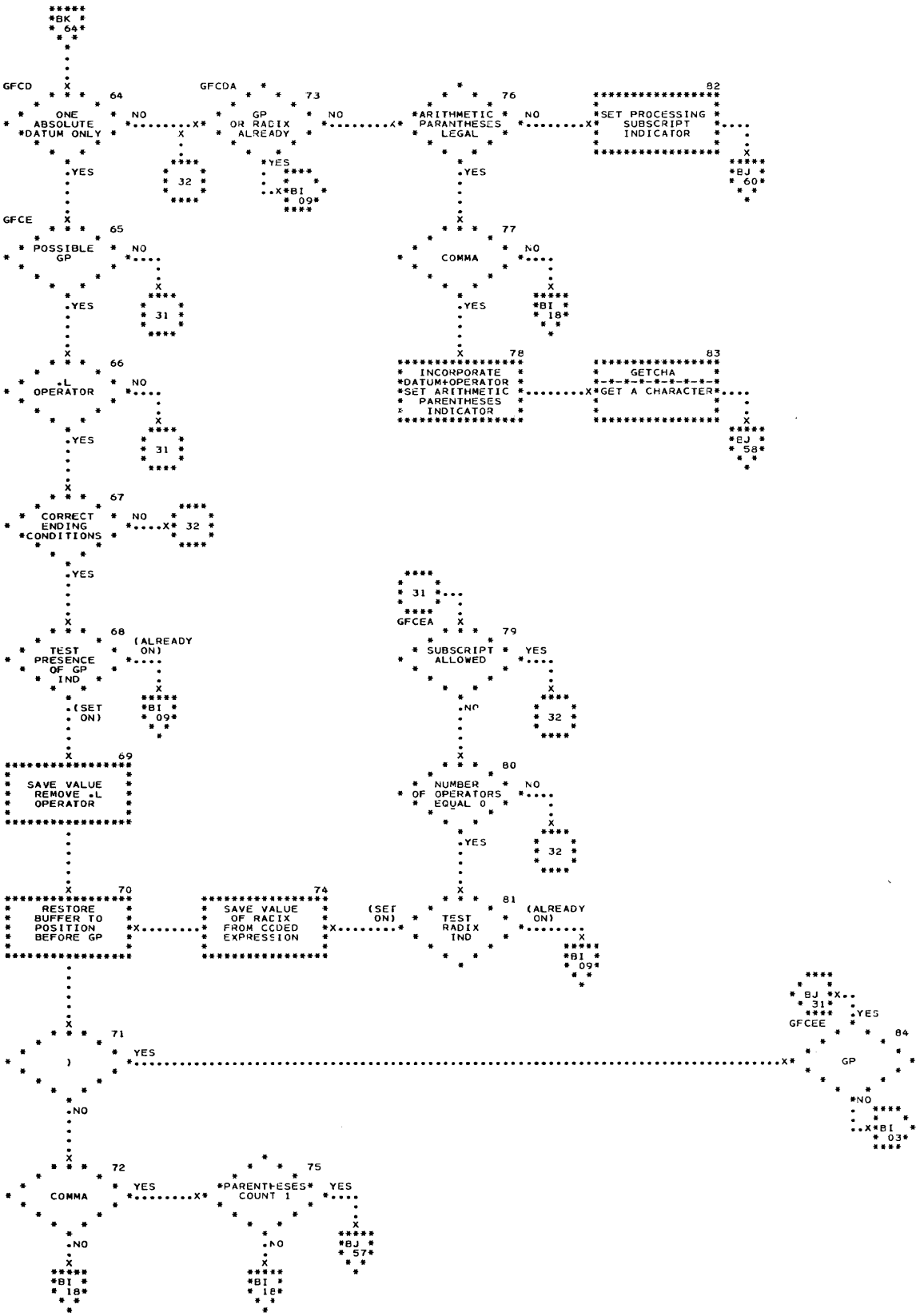
However, when a comment begins in the statement field, GETCHA not only inserts each character into the collection buffer for the statement characters as usual, but also gives each character to main flow. MAIN then ignores the characters in a statement comment by relooping into GETCHA until a semicolon appears in $3.

Boxes 5, 7, 8, 12, 13, 18, 19, 25, 26

The specifications for including a coprocessor subroutine with Pass 1 is described in a later section of this manual. During her initialization GETCHA determines whether there is a coprocessor subroutine operating with Pass 1 and fills in GETCHA's error branch vector accordingly. When there is no coprocessor subroutine with STRAP II or when a coprocessor subroutine does not want control for one of its error conditions, then GETCHA or MAIN gives an error message and continues with the appropriate corrective action.


BI 18
1
GETFLD
INITIAL
HOUSEKEEPING
9
RESTORE BUFFER TO ORIGINAL CONDITION, MAKE NULL CODED EXPRESSION
18
ERROR MESSAGE
GFCBIN
2
HOUSEKEEPING FOR ( )
10
ERROR MESSAGE
BI 02
3
1ST CHARACTER OBTAINED
YES
NO
11
COMMA SEMICOLON OR QUOTE
YES
NO
19
EXIT
YES
BI 03
GFMAIN
4
GETCHA
GET A CHARACTER
12
( )
YES
NO
20
PARENTHESES COUNT EQUAL ZERO
NO
BI 04
5
( )
YES
NO
13
( )
NO
YES
21
GETCHA
GET A CHARACTER
BI 05
6
SET SUBSCRIPT-NOT-ALLOWED INDICATOR
14
ADD 1 TO PARENTHESES COUNT
7
START OF SYMBOL OR $
YES
NO
15
CPLTSY
COMPLETE A SYMBCL
GFMAJ
22
SYSTEM SYMBOL
NO
YES
25
$ SYMBOL
YES
NO
28
PREFIX AND LENGTH OF SYMBOL IN CODED EXPRESSION
GFMB
8
START OF NUMBER
YES
NO
16
CPLTNM
COMPLETE A NUMBER
23
PROPER SYSTEM SYMBOL
NO
YES
26
PREFIX AND LENGTH OF SYMBOL IN CODED EXPRESSION
29
ADD + OPERATOR TO CODED EXPRESSION
BJ 30
17
VALUE, PREFIX, AND DDS TO CODED EXPRESSION
24
VALUE, PREFIX, AND DDS TO CODED EXPRESSION
27
SET INDEX-SUBSCRIPT-OK INDICATOR ON
BJ 58
BJ 58
BJ 58
BJ 58
BJ 5E


*BJ * * 30*
GFMC
30
IS CHARACTER + - . * / *
YES
NO
39
SET PROPER OPERATOR CODE IN CODED EXPRESSION
47
GETCHA
GET A CHARACTER
52
SET OPERATOR PRESENT INDICATOR
*BJ * * 58*
GFMAF
58
ADD 1 TO COUNT OF DATUM SET TO SHOW END OF DATUM
GFMH
31
SYNTAX CORRECT
NO
YES
40
NULL FIELD
NO
YES
*BI * * 09*
*BI * * 09*
53
SYNTAX CORRECT
NO
YES
59
IF LAST OPERATR WAS .L. CHANGE OPERATOR TO .+.
*BJ * * 31*
32
PULL UP LIST
41
NULL FIELD INDICATOR ON
54
SET NEW SYNTAX
33
PAREN COUNT 0
NO
YES
42
COMMA
NO
YES
*BI * * 18*
NO
48
)
YES
55
GFLONG
CHECK FOR ROOM IN BUFFER
*BI * * 05*
*BJ * * 60*
34
NULL FIELD
NO
YES
43
GFZA
PUT OPERATORS IN CODED EXPRESSION
56
ALREADY ONE SUBSCRIPT
YES
NO
*BK * * 64*
GFCCA
60
GFZA
PUT OPERATORS IN CODED EXPRESSION
35
CREATE NULL PREFIX AND DATUM
44
ARITHMETIC PARENTHESES
NO
49
SHOW
EXISTENCE OF SUB-CODED EXPRESSIONS
SET END OF SUB-CODED EXPRESSION INDICATOR
36
IS GP ALLOWED
NO
YES
45
GP WAITING
YES
NO
50
GETCHA
GET CHARACTERS TILL PROPER END CHARACTER
62
COMMA
YES
NO
37
GP WAITING
NO
YES
46
SET CCDED EXPRESSION TO SHOW NO GP
EXIT
63
)
NO
YES
*BJ * * 57*
GFCA
57
ALREADY ONE SUBSCRIPT
NO
YES
*BI * * 04*
*BI * * 18*
38
SET VALUE OF GP IN CODED EXPRESSION
51
ALREADY GP OR RADIX
YES
NO
*BI * * 09*
*BI * * 02*
*BI * * 02*
*BI * * 02*


BK 64
GFCD
64 ONE ABSOLUTE DATUM ONLY
NO
YES
32
GFCDA
73 GP OR RADIX ALREADY
NO
YES
BI 09
76 ARITHMETIC PARANTHESES LEGAL
NO
YES
82 SET PROCESSING SUBSCRIPT INDICATOR
BJ 60
GFCE
65 POSSIBLE GP
NO
YES
31
77 COMMA
NO
YES
BI 18
66 .L OPERATOR
NO
YES
31
78 INCORPORATE DATUM+OPERATOR SET ARITHMETIC PARENTHESES INDICATOR
83 GETCHA GET A CHARACTER
BJ 58
67 CORRECT ENDING CONDITIONS
NO
32
YES
GFCEA
31
79 SUBSCRIPT ALLOWED
YES
NO
32
68 TEST PRESENCE OF GP IND
(ALREADY ON)
(SET ON)
BI 09
80 NUMBER OF OPERATORS EQUAL 0
NO
YES
32
69 SAVE VALUE REMOVE .L OPERATOR
70 RESTORE BUFFER TO POSITION BEFORE GP
74 SAVE VALUE OF RADIX FROM CCDED EXPRESSION
(SET ON)
81 TEST RADIX IND
(ALREADY ON)
BI 09
GFCEE
BJ 31
YES
84 GP
NO
BI 03
71 )
YES
NO
72 COMMA
YES
NO
BI 18
75 PARENTHESES COUNT 1
YES
NO
BI 18
BJ 57

LINKAGE

On Entry

1. The calling sequence is:
SIC, Getflz; B, GETFLD.
2. Since the index words, Bcfldx and Bcfxtx, define the buffer area where the coded expression is to be built up,
   a. the value field of Bcfldx locates the full word address of the buffer and
   b. the value field of Bcfxtx locates the first bit after the buffer.
3. The bit, Gfchar, is on if the user has already set up $3 with the first character of the field.
4. The bit, Gfnogp, is on if the user does not allow a general parenthetical integer entry on the field about to be encoded.

On Exit

1. The value field of Bcfldx will have been updated accordingly.
2. The value field of $3 will contain the first character after the field.
3. The flag, Fsymb, will be on if a symbol (which was not an absolute system symbol) appeared in the field.

USER(s)

The most frequent use of GETFLD is from the place in main flow where MAIN calls on GETFLD to put into the unit the encoding of the address field(s) from an instruction that produces binary output. (See Appendix C.) However, instructions that are re-routed around this area of MAIN for special treatment in one of the sub-procedures of MAIN have their address fields encoded (If necessary) from that particular subroutine. Subroutines using GETFLD include: MQM, MDIMRF, MDIMRT, MIOD, MDUP, MX, MQS. MAIN also uses GETFLD during the processing of the byte size and field length.

DETAILS

A coded expression for one field will be set up in the beginning of the buffer. The end of the buffer may have been used for intermediate storage. Moreover, if the value field of Bcfxtx does not contain a full word address, the address will be rounded down.

Each coded expression which GETFLD is called upon to make is usually saved in the current expanded instruction unit. During Pass 1 the coded expression for each field of an instruction producing binary output is sequentially stacked in the variable length portion of the current unit. During Pass 2 these coded expressions will be sequentially decoded and evaluated by the routine DECODE and its VALUE subroutine in order to obtain the values to be inserted in the binary output. (The binary output for an instruction is also saved in the unit.)

GETFLD performs a legality test to detect syntactical errors in a field. If the combination of datums is specified in an illegal manner, GETFLD encodes a null coded expression which will be evaluated later as zero for the field.

A code number is associated with the type of information that either precedes or follows the arithmetic separators: (, ., +, -, *, /, ). An S-code (syntax code) is a number associated with what precedes a separator; a syntax code describes either a datum or another arithmetic separator. A U-code is a number associated with what follows an arithmetic separator; a U-code describes either a datum or another arithmetic separator, one separator or two separators can determine an arithmetic operation.

With the S- and U-codes, GETFLD looks up an element in its two dimensional syntax matrix. Each look up in the syntax matrix accomplishes two things: First, the look up establishes whether or not the current operation, or the current datum is syntactically correct depending on the value of the isolated element. Second, the look-up establishes the S-code, the value of the isolated element to be used for the next look-up in the syntax matrix. Initially each S- and U-code is zero. Thereafter, each U-code is determined by the current datum or by the current arithmetic separator; each S-code is determined by the previous look-up in the syntax mateix.

Syntax Codes (what precedes)

| Value(s) | Symbol | Meaning |
|---|---|---|
| 0 | B | Beginning of expression, or unitary minus. |
| 1 | S | Symbol, number, or parameter. |
| 2 | SZ | Symbol etc. preceded by a decimal. |
| 3 | O | Operation. |

| Value(s) | Symbol | Meaning |
|---|---|---|
| 4 | . | . preceded by an 'S' described above. |
| 5 | .Z | . not preceded by 'S'. |

U-Codes

| Value(s) | Meaning |
|---|---|
| 0 | Symbol, number, or parameter |
| 1 | + and - operations |
| 2 | * , / or END of expression |
| 3 | . |
| 4 | Special to identify arithmetic ( ) |

FLOW CHART DESCRIPTION

Box 1

Initial housekeeping for GETFLD is done here. Bcfxtx (last word available in the intermediate buffer) is rounded down to a full word address. Gfpcnt (parentheses count) is set to zero. Gfnull and Gfgenp are turned off. The position within the intermediate buffer is stored in Bcfldz. $2 is spaced over the first prefix bit.

Box 2

The first word of the GETFLD buffer (Gfsave) is set to zero and the buffer is pushed down one word. If there is a special radix (Gfrads on), its value is stored in Gfsrad.

Boxes 3, 4

If the first character has not been obtained, the program branches to GETCHA.

Box 5

If the character is a parentheses, Gfgenp, Gfrads, Gfsfrd, Gfsfxa and Gfsfxb are set to zero. The parenthesis count is stepped one. If the parenthesis is possibly arithmetic, Bcfldx is spaced back three bits, removing the prefix. Gfapan is set to one.

Boxes 6-8

Gfsfsb (subscript-allowed indicator) is turned off. The character is checked to determine if it is either a($), or the start of a symbol, or the start of a number.

Box 15

If the character is either a($) or the start of a symbol, 10 bits are left in the coded expression for the prefix and length of symbol, and CPLTSY gets the entire symbol.

Boxes 22-24

If it is a properly defined system symbol the intermediate buffer is spaced back 10 bits. The prefix, value and dds of the symbol are placed in the buffer, and the buffer is spaced up 43 bits.

Boxes 25-27

If it is not the($) symbol, the prefix and length of symbol are stored in the coded expression, and the subscript allowed indicator is turned on.

Boxes 28, 29

The symbol is the($). The prefix and value of the($) is stored in the coded expression. The buffer is spaced 20 bits.

Boxes 16, 17

The character is the start of a number. CPLTNM furnishes the number. Gfrads is turned off. The prefix, datum, and data description are stored in the encoded expression, and the buffer pointer is spaced twenty-eight bits.

Boxes 30, 39, 47, 52

The character is tested for(+),(-),(.),(*), or(/). If one of these, theappropriate operator code and the number of the datum acted on by the operator are stored in the coded expression; the syntax U-code indicating what may follow is set; Gfspar (operator-present indicator) is set to one; one is added to the operator count. GETCHA is asked for the next character.

Boxes 53-55

The U- and S-codes are checked for syntactical error. If correct, the new S-code is saved and the routine branches to Gflong to check for sufficient room in the intermediate buffer.

Boxes 31, 32, 40, 41

The syntax is checked for the legality of ending at this point. If the syntax is legal or if it is a null

field, the GETFLD buffer is pulled up. If null, the null field indicator (Gfnull) is turned on.

Boxes 34, 35

If the parentheses count is zero, and Gfnull is on, the null prefix is stored in the intermediate buffer, and $2 is spaced six bits.

Boxes 36, 45, 50

If a GP is not allowed (indicated by Gfnogp on), and a GP does not follow (Gfgenp off), exit is made. If a GP does follow, GETCHA is requested to space to the end of the instruction unit.

Boxes 37, 46

If a GP is allowed, but no GP follows, the bit at the end of the coded expression is set to zero to indicate no GP entry exit.

Boxes 38

If a GP follows, the heading for a GP entry is stored in the buffer. $2 is spaced 8 bits, and the 1st word if the GETFLD buffer reset to zeros.

Boxes 42,48

If the parentheses count is not zero (box 33), the null field indicator is turned off. If the character is not a , and ), the error message (Inappropriate Character) is given.

Boxes 56, 60

If the character is a , or ) and a subscript has already been encounter (Gfsfxb on), a branch is made to Gfza. In the case of a ), one is subtracted from the parenthesis count. Gfza stores the operators into the coded expression from the GETFLD buffer. The expression is given the prefix 1000.

Box 61

Set coded expression to indicate no more operators $2 is spaced 1 bit.

Boxes 62, 57, 51

The character is checked for a comma indicating a subscript. Bit Gfsfxb, if on indicates a subscript has already been processed and the program branches to process the subscript. If off, a check is made on bit Gfsfxa. If off, a GP or radix has not been encountered, and the bit is turned on. If on, the error message (Syntax Error) will be given.

Box 63

If the test for a comma (box 62) is negative, the character is checked for a closed parenthesis. If yes, an error message is given. If no, the program will request the next character from GETCHA.

Boxes 43, 44, 49

If the field is not null (box 34), the program branches to Gfza. If there was no operator to make the parentheses arithmetic, the coded expression is set to indicate a sub-expression.

Boxes 64, 73

If there is only one absolute datum, and a GP or radix has already been processed (Gfsfxa on), an error message is given.

Boxes 76-78, 82,83

If the parenthesis could not be arithmetic, Gfsfxb (processing-subscript indicator) is turned on and the program branches to Gfza. If the parentheses could be arithmetic, but the character is not a closed parenthesis, the error message (Inappropriate Character) is given. If a closed parenthesis, the operator and the number of the datum are placed in the GETFLD buffer. GETCHA supplies the next character and the opening arithmetic parenthesis; indicator (Gfspas) is turned on.

Boxes 65, 66, 79-81, 74

The expression is tested for the possibility of being a GP or radix. If neither, Gfsfsb (subscript-allowed indicator) is tested. If off, and the number of operators is equal to zero, the radix indicator (Gfrads) is turned on. If it is already on, the error message (Syntax Error) is given.

Box 74

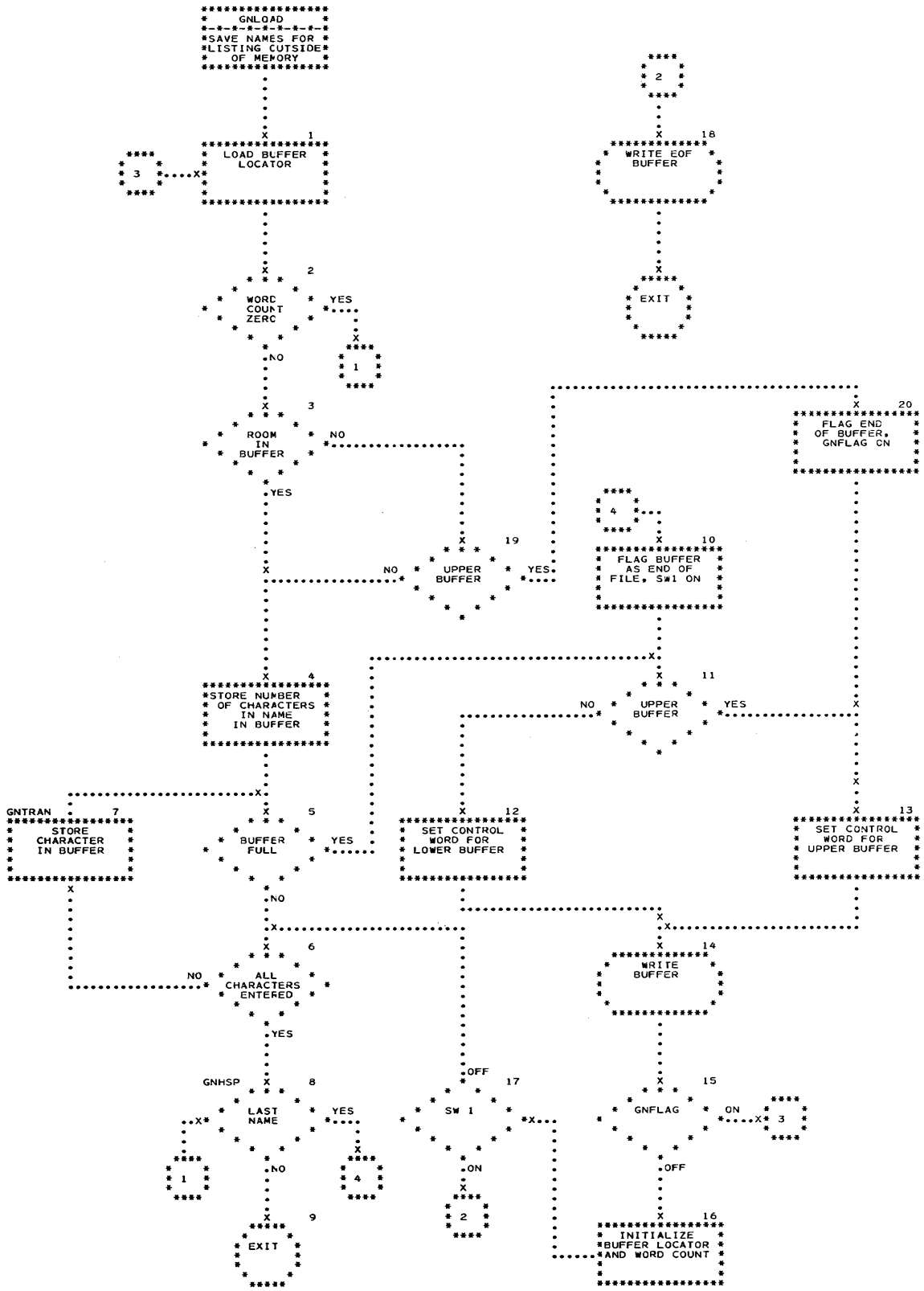
The value of the radix is stored in Gfsrad.

Boxes 67-70

If the syntax is correct for terminating and a GP has not already been processed, Gfgend is turned on. The value of the GP is stored in Gfgpv, and the operator is deleted from the GETFLD buffer.

Box 71

If the character is neither a closed parenthesis, nor a comma, an error message is given. If a closed parenthesis, it is checked for a GP. If a comma, and the parenthesis count is not one, an error message is given.


GNLOAD
SAVE NAMES FOR LISTING OUTSIDE OF MEMORY
1
LOAD BUFFER LOCATOR
2
WORD COUNT ZERO
YES
NO
3
ROOM IN BUFFER
NO
YES
4
STORE NUMBER OF CHARACTERS IN NAME IN BUFFER
5
BUFFER FULL
YES
NO
6
ALL CHARACTERS ENTERED
NO
YES
GNTRAN
7
STORE CHARACTER IN BUFFER
GNHSP
8
LAST NAME
YES
NO
9
EXIT
10
FLAG BUFFER AS END OF FILE, SW1 ON
11
UPPER BUFFER
NO
YES
12
SET CONTROL WORD FOR LOWER BUFFER
13
SET CONTROL WORD FOR UPPER BUFFER
14
WRITE BUFFER
15
GNFLAG
ON
OFF
16
INITIALIZE BUFFER LOCATOR AND WORD COUNT
17
SW 1
ON
OFF
18
WRITE EOF BUFFER
19
UPPER BUFFER
NO
YES
20
FLAG END OF BUFFER, GNFLAG ON
EXIT

## PURPOSE

GNLOAD is the subroutine which creates the name file for STRAP II and writes it on the disk. It uses two buffers, each being written on the disk as it is filled.

## LINKAGE

### On Entry

The calling sequence is:

1. SIC, Gnexit; B, GNLOAD.
2. $7 must contain the location of the name in the value field, and the number of name characters in the count field. The file is terminated when the count field is found to be zero.

### On Exit

GNLOAD places the name specified by the calling sequence in its buffer with the following format:

Each name record

| | |
|---|---|
| 1st character | 8-bit byte, number of A8 characters in the name + one for the check character. |
| 2nd character | 8-bit byte, check character for the name entry, progresses from 0-9 and repeats. |
| next n characters | 8-bit bytes, name in A8 code. Names less than eight characters in length are assumed to be eight characters long. |
| end-of-buffer character | The end of the second buffer is terminated by an 8-bit byte of all ones. |
| end-of-file character | The file is terminated by an 8-bit byte of all zeros. |

## FLOW CHART DESCRIPTION

### Boxes 1, 2

The count field of index register 7 is checked for zero. If zero, end-of-file is initiated. If not zero, the name will be entered into the buffer.

### Boxes 3, 4

A check is made to determine if there is sufficient room remaining in the current buffer for the name. If so, the number of characters in the name is stored in the buffer.

### Boxes 5-7

A check is made to determine if the buffer is full. If not, each character of the name is stored in the buffer in 8-bit bytes.

### Boxes 8, 10

If the flag is on in index register 7 the current name is the last, and the file is terminated by storing a character count of zero in the buffer, and switch 1 is turned on.

### Boxes 11-14

The completed buffer is written on the disk.

### Box 15

If the Gnflag is on indicating that there had not been room in the second buffer, the flag is turned off and return is made to store the name in the first buffer.

### Box 16

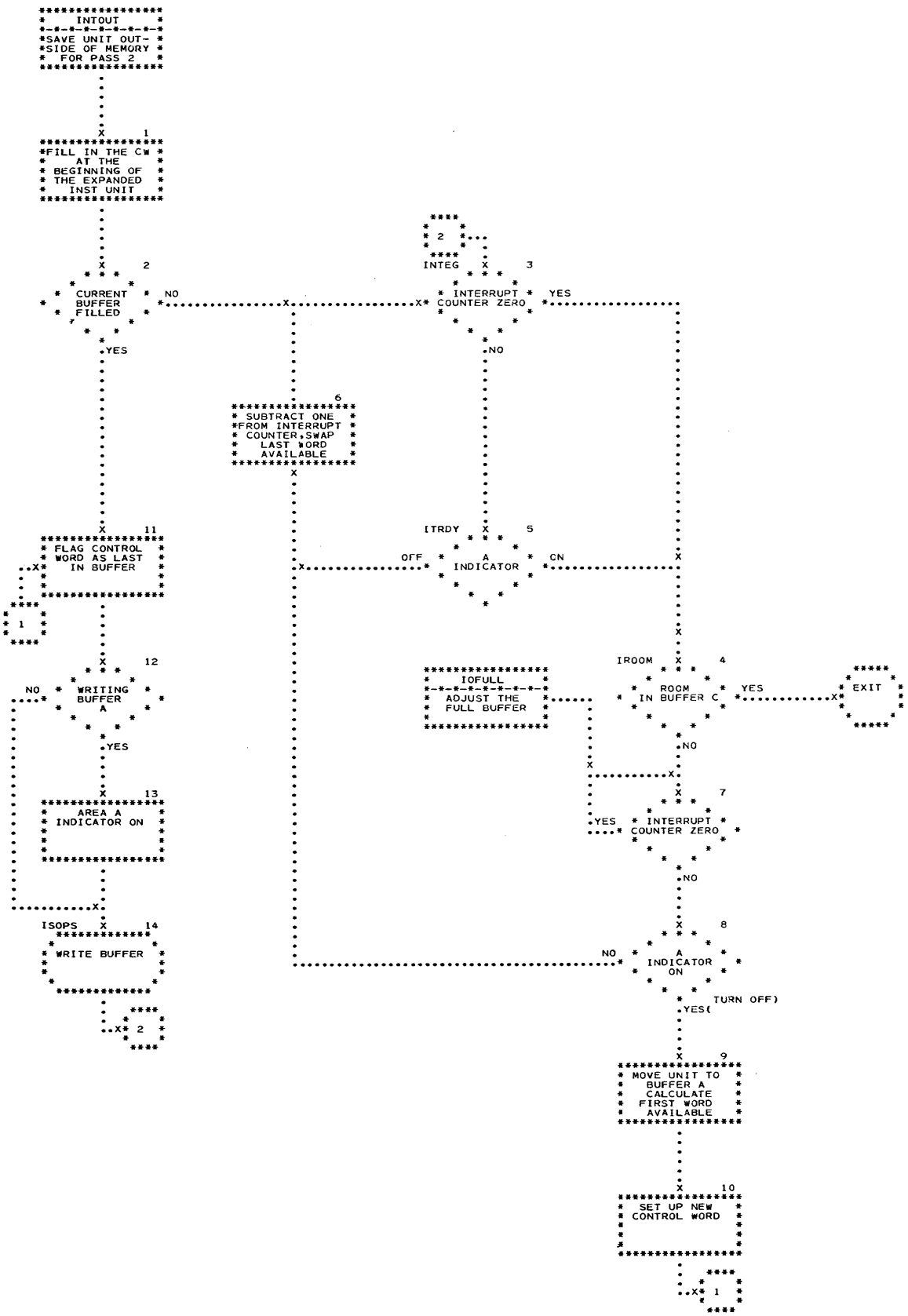
Initialize buffer locator and counter to beginning of buffer.

### Boxes 17, 18

Switch 1, if on, indicates last buffer has been processed, and a terminating arc is written as end-of-file.

### Boxes 19, 20

If processing in the second buffer, the end of the buffer is flagged with an 8-bit byte of all one bits.


INTOUT
SAVE UNIT OUT- SIDE OF MEMORY FOR PASS 2
1
FILL IN THE CW AT THE BEGINNING OF THE EXPANDED INST UNIT
2
CURRENT BUFFER FILLED
NO
YES
INTEG
3
INTERRUPT
YES
COUNTER ZERO
NO
6
SUBTRACT ONE FROM INTERRUPT COUNTER, SWAP LAST WORD AVAILABLE
11
FLAG CONTROL WORD AS LAST IN BUFFER
ITRDY
5
OFF
A INDICATOR
ON
IROOM
4
ROOM IN BUFFER C
YES
EXIT
12
WRITING BUFFER A
NO
YES
IOFULL
ADJUST THE FULL BUFFER
13
AREA A INDICATOR ON
7
INTERRUPT
YES
COUNTER ZERO
NO
ISOPS
14
WRITE BUFFER
8
A INDICATOR ON
NO
TURN OFF)
YES(
9
MOVE UNIT TO BUFFER A CALCULATE FIRST WORD AVAILABLE
10
SET UP NEW CONTROL WORD

## PURPOSE

INTOUT buffers the expanded units and writes them on the disk for processing during Pass 2.

## LINKAGE

1. The calling sequence is: SIC, Intouz; B, INTOUT.
2. The index word, Ibuffw, must point to the next available full word location in the unit buffer.
3. When sufficient room is not available for the unit in the current buffer, the calling sequence is: SIC, Intouz; B, INTOUT.

## DETAILS

Each expanded unit is preceded by a control index word having the format shown in Figure 15.

The routine uses three buffers (A, B, and C), each being written on the disk as it is filled. The location in Ibuffw is the location of the next word available to be used by INTOUT or the user routine.

## FLOW CHART DESCRIPTION

### Box 1

The number of words in the unit is obtained by subtracting the address of the last control-word Icwa from the address of the next word available Ibuffw, and is placed in the count field of the control word. The address of the new control word will be the address in Ibuffw, and this is stored in the refill field. Ibuffw is stepped by one to indicate next word available.

### Box 2

A check is made to determine if Ibuffw has left the current buffer by comparing it with the address of the last word of the current buffer plus one.

### Boxes 3, 4

Each time an EOP interrupt occurs due to the completion of the write operation, one is added to the interrupt counter Intcnt. This count is checked here to see if it is equal to zero. If it is equal to zero, a check is made to see if at least ten unused words remain in buffer C. If so, return is made.

### Box 5

The area A indicator is turned on at the beginning of writing buffer A, and turned off when writing buffer C. It's being off indicates that the last buffer written was C, and that the last word available Ibulwa must be updated.

### Box 6

One is subtracted from the interrupt counter, and the last word available table is rotated once.

### Box 7

If the interrupt counter is zero, the program waits until an EOP interrupt occurs.

### Boxes 8-10

If the area A indicator is on at this point, it is turned off, the incomplete unit is moved to buffer

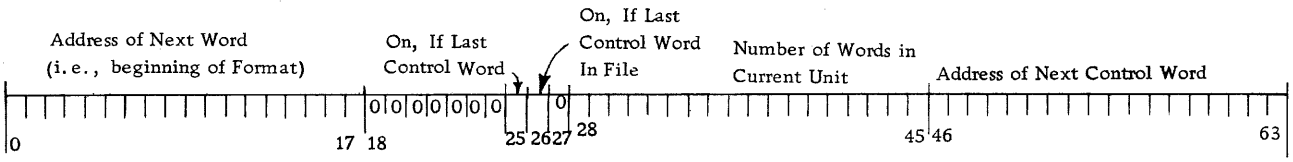



**Figure 15. Format of the First Word in a Unit**

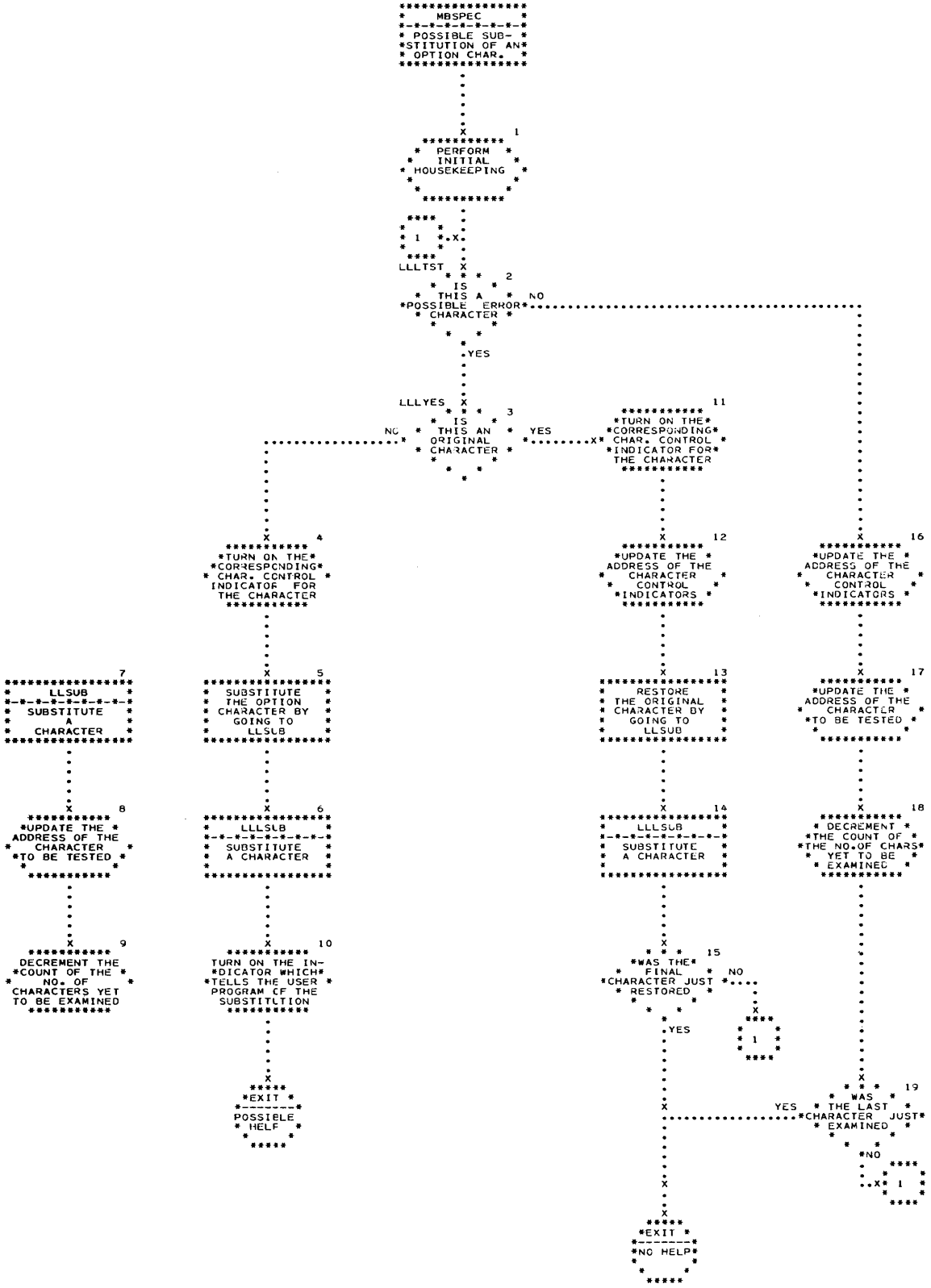
A, the refill field of the last control word is updated to reflect this, and a new control word is set up in buffer A.

Box 11

Bit 25 of the last control word is flagged to indicate last control word in buffer.

Boxes 12-14

If buffer A has just been completed, the area A indicator is turned on. The completed buffer is written on the disk.


MBSPEC
POSSIBLE SUB- STITUTION OF AN OPTION CHAR.
PERFORM INITIAL HOUSEKEEPING
1
LLLTST
IS THIS A POSSIBLE ERROR CHARACTER
2
NO
YES
LLLYES
IS THIS AN ORIGINAL CHARACTER
3
NO
YES
TURN ON THE CORRESPONDING CHAR. CONTROL INDICATOR FOR THE CHARACTER
11
UPDATE THE ADDRESS OF THE CHARACTER CONTROL INDICATORS
12
RESTORE THE ORIGINAL CHARACTER BY GOING TO LLSUB
13
LLLSUB
SUBSTITUTE A CHARACTER
14
WAS THE FINAL CHARACTER JUST RESTORED
15
NO
YES
1
EXIT NO HELP
UPDATE THE ADDRESS OF THE CHARACTER CONTROL INDICATORS
16
UPDATE THE ADDRESS OF THE CHARACTER TO BE TESTED
17
DECREMENT THE COUNT OF THE NO. OF CHARS YET TO BE EXAMINED
18
WAS THE LAST CHARACTER JUST EXAMINED
19
NO
YES
1
TURN ON THE CORRESPONDING CHAR. CONTROL INDICATOR FOR THE CHARACTER
4
SUBSTITUTE THE OPTION CHARACTER BY GOING TO LLSUB
5
LLLSUB
SUBSTITUTE A CHARACTER
6
TURN ON THE INDICATOR WHICH TELLS THE USER PROGRAM OF THE SUBSTITUTION
10
EXIT POSSIBLE HELP
LLSUB
SUBSTITUTE A CHARACTER
7
UPDATE THE ADDRESS OF THE CHARACTER TO BE TESTED
8
DECREMENT THE COUNT OF THE NO. OF CHARACTERS YET TO BE EXAMINED
9

## LINKAGE

On Entry

1. The calling sequence is:

```
SIC, $15; B, MBSPEC
, address of the test word (I)      ' no further help
B, a fix-up subroutine                possible return
Normal Return                       ' possible help re-
                                      turn
```

2. A string of 8 MBSPEC indicators, Llbts, must be preset to zero by the user program before MBSPEC works on an op or system symbol.

3. The op or system symbol (i.e., any test word) must be eight A8 characters in length.

On Exit

1. Whenever MBSPEC makes a substitution, it turns on a special indicator, Llbts.8, for the user program as well as exiting through the possible help return. (MBSPEC turns this indicator off in its initial housekeeping.)

2. When MBSPEC exits through the no further help possible return, the op or system symbol is in its original illegal form.

## FLOW CHART DESCRIPTION

### Box 1

Initialization includes turning of a substitution-has-been-made indicator, setting the indices to locate the starting location of the test word and of the eight control indicators, and resetting the counter.

### Box 2

During Pass 1 after the main flow has discovered that the second or any subsequent character of an op or of a system symbol is illegal, Pass 1 goes to the MBSPEC to see if a (0) was punched for an (O), a (1) for an (I), a (2) for a (Z), a (5) for an (S), or vice versa on the original input. More characters can be included in the possible error character set by adjusting two tables of MBSPEC.

### Boxes 3, 4, 11

MBSPEC maintains a string of eight consecutive indicators, each one represents the status of its corresponding character in the test word. MBSPEC uses and sets these indicators to determine whether each character in an op or system symbol is the original character or the substitute character: 0 if the character is the original character; 1 if the character is the newly substituted option character. Pass 1 zeroes these indicators before the first entry into MBSPEC. But during the ping-ponging back and forth while MBSPEC is supplying Pass 1 with a new substitute test word, these eight control indicators remain as last set by MBSPEC.

### Box 6

Let each character in the test word be a possible error character. If we represent an original character in the test word by an X and a substitute character by an S, the following list represents the order of the possible test word given to the using program by MBSPEC:

SXXXXXXX, XSXXXXXX, SSXXXXXX,
XXSXXXXX, etc.

### Boxes 7-9

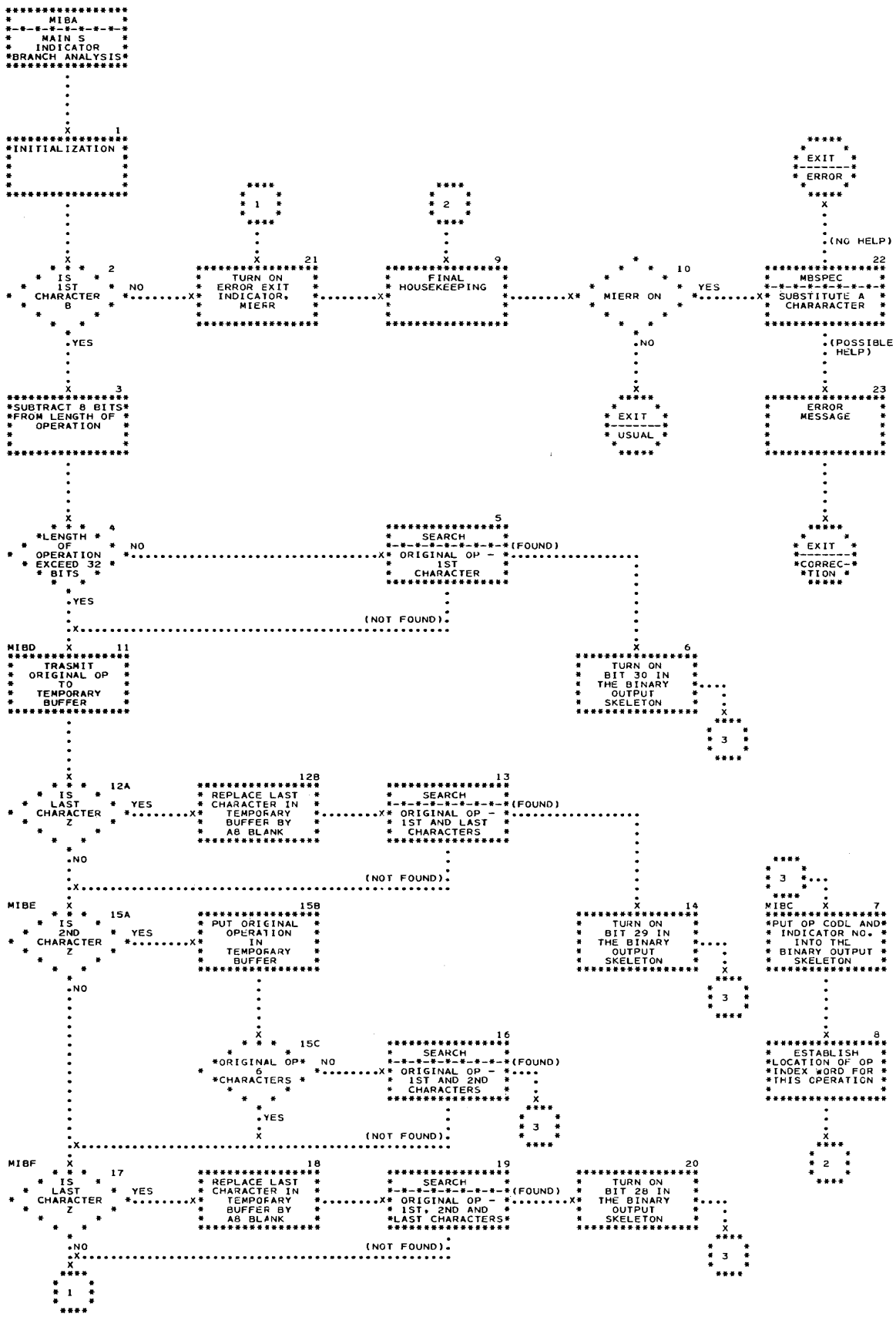
The updating and incrementing are only significant if the option character is being replaced by the original character.

### Box 10

Whenever main flow makes a successful search on the op table or system symbol table, main flow tests the indicator, Llbts.8. If it is on, main flow will give an error message informing the programmer a change has been made to his original input.

### Boxes 15, 19

Once MBSPEC returns to the using program by its no help exit, the using program ceases coming to MBSPEC for help on the particular illegal op or system symbol.


MIBA
MAIN S INDICATOR BRANCH ANALYSIS
INITIALIZATION
IS 1ST CHARACTER B
NO
YES
TURN ON ERROR EXIT INDICATOR, MIERR
FINAL HOUSEKEEPING
MIERR ON
YES
NO
EXIT USUAL
EXIT ERROR
(NO HELP)
MBSPEC SUBSTITUTE A CHARARACTER
(POSSIBLE HELP)
ERROR MESSAGE
SUBTRACT 8 BITS FROM LENGTH OF OPERATION
LENGTH OF OPERATION EXCEED 32 BITS
NO
YES
SEARCH ORIGINAL OP - 1ST CHARACTER
(FOUND)
(NOT FOUND)
EXIT CORREC-TION
MIBD
TRASMIT ORIGINAL OP TO TEMPORARY BUFFER
TURN ON BIT 30 IN THE BINARY OUTPUT SKELETON
IS LAST CHARACTER Z
YES
NO
REPLACE LAST CHARACTER IN TEMPORARY BUFFER BY A8 BLANK
SEARCH ORIGINAL OP - 1ST AND LAST CHARACTERS
(FOUND)
(NOT FOUND)
MIBE
IS 2ND CHARACTER Z
YES
NO
PUT ORIGINAL OPERATION IN TEMPORARY BUFFER
TURN ON BIT 29 IN THE BINARY OUTPUT SKELETON
MIBC
PUT OP CODL AND INDICATOR NO. INTO THL BINARY OUTPUT SKELETON
ORIGINAL OP* CHARACTERS * 6
NO
YES
SEARCH ORIGINAL OP - 1ST AND 2ND CHARACTERS
(FOUND)
(NOT FOUND)
ESTABLISH LOCATION OF OP INDEX WORD FOR THIS CPERATION
MIBF
IS LAST CHARACTER Z
YES
NO
REPLACE LAST CHARACTER IN TEMPORARY BUFFER BY A8 BLANK
SEARCH ORIGINAL OP - 1ST, 2ND AND LAST CHARACTERS
(FOUND)
(NOT FOUND)
TURN ON BIT 28 IN THE BINARY OUTPUT SKELETON
1
2
3
4
5
6
7
8
9
10
11
12A
13
14
15A
15B
15C
16
18
19
20
21
22
23
128
158

LINKAGE

On Entry

1. MAIN branches directly to MIBA.
2. The complete operation mnemonic is in the operation collection buffer, Bop.

On Exit

1. The proper indicator number, indicator operation code, and indicator bits are set up in the binary output skeleton contained in the expanded instruction unit.
2. The location of this op's index word is established.
3. MIBA returns to one of three locations in MAIN; The three subsequent actions that can occur in MAIN after MIBA has analyzed the op are:
   a. To continue processing the branch-on-indicator operation.
   b. To relook-up in the primary operation table the formerly illegally specified operation mnemonic which MIBA has now adjusted with the MBSPEC subroutine.
   c. To treat an illegal operation which cannot be successfully adjusted.

FLOW CHART DESCRIPTION

Boxes 1, 9

MIBA uses an index register which has a standard use in main flow. This index register must be saved while MIBA is in process and restored before MIBA enters main flow.

Boxes 2, 9, 10, 21-23

If the op does not start with the character, B, or is not a legally specified BI-type op, the operation will receive the STRAP error op treatment here in MIBA. The adjustment of the op mnemonic may make it possible for MAIN to find the op in another search of the primary op table.

Boxes 3-5; 3,4, 11-13; 3,4, 11,12,15,16; 3,4,11,12,15,17-19

For each of the indicators that can appear in the BI-type instruction, MIBA has in its table one entry with the mnemonic for the indicator and its associated indicator number (See Appendix A.)

The variations of a BI-type of instruction are: BI, BIZ, BZI, and BZIZ. The maximum mnemonic for an indicator contains four characters.

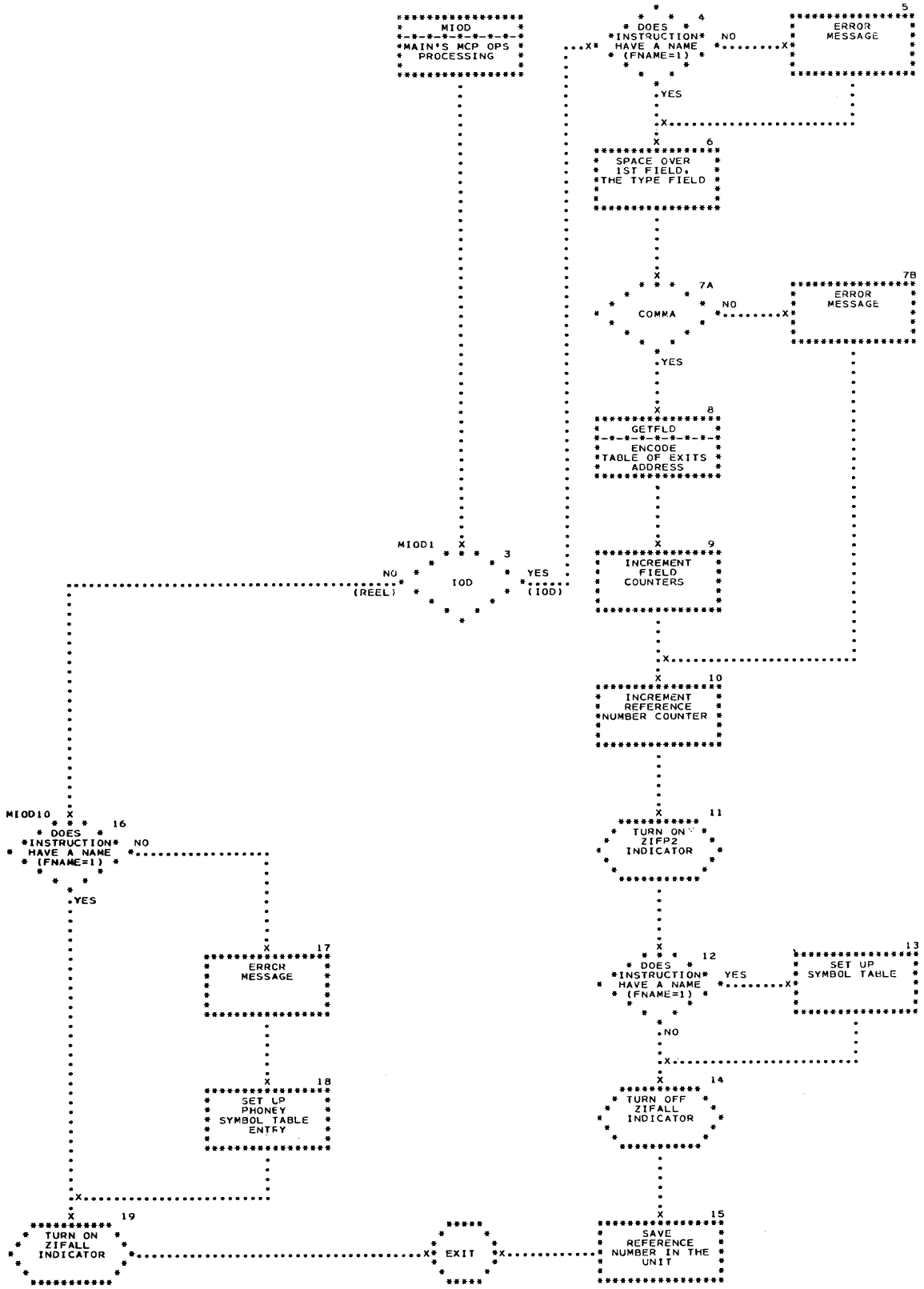
By successive testing of the last and second characters of the original operation mnemonic for a Z and by subsequent selected stripping of the basic operation mnemonic, MIBA tests to see if any one of the resulting indicator mnemonics is in the table of the indicator mnemonics.

Boxes 6, 14, 20, 7

The binary output skeleton for the instruction is contained in the fixed format portion of the op which is in the expanded instruction unit. MIBA inserts in the binary output skeleton the indicator number, the operation code, and the setting for bits 29 and 30. (The fixed portion of each new unit is zeroed during the processing initialization for each new instruction.)

Box 8

MIBA also supplies main flow with the information that is usually obtained from the primary operation table. This location of the op index is put into the unit and into the value field of $X5 for referencing purposes.


MIOD
MAIN'S MCP OPS PROCESSING
MIOD1
3
IOD
NO (REEL)
YES (IOD)
4
DOES INSTRUCTION HAVE A NAME (FNAME=1)
NO
YES
5
ERROR MESSAGE
6
SPACE OVER 1ST FIELD, THE TYPE FIELD
7A
COMMA
NO
YES
7B
ERROR MESSAGE
8
GETFLD ENCODE TABLE OF EXITS ADDRESS
9
INCREMENT FIELD COUNTERS
10
INCREMENT REFERENCE NUMBER COUNTER
11
TURN ON ZIFP2 INDICATOR
12
DOES INSTRUCTION HAVE A NAME (FNAME=1)
YES
NO
13
SET UP SYMBOL TABLE
14
TURN OFF ZIFALL INDICATOR
15
SAVE REFERENCE NUMBER IN THE UNIT
MIOD10
16
DOES INSTRUCTION HAVE A NAME (FNAME=1)
NO
YES
17
ERROR MESSAGE
18
SET UP PHONEY SYMBOL TABLE ENTRY
19
TURN ON ZIFALL INDICATOR
EXIT

## LINKAGE

The MCP ops, IOD and REEL, have a special bit on in their op index word, signifying the op is an MCP op. Just before the digit select table in Pass 1 there is a test of this bit. If it is on, the address fields of the current op will then be processed by MIOD and not by the main routine. If the current op index has this bit off, or if a comment, then MAIN turns on the Fiod indicator to show that there should not be any more MCP-type instructions and continues as usual. Thus Fiod should not be on while MIOD is processing a REEL or an IOD instruction.

## FLOW CHART DESCRIPTION

### Boxes 4-15

The spacing over the first field is accomplished by saving the unit buffer control address and then after asking GETFLD to encode the type field, resetting the buffer area to the original address. The encoding is thus effectively erased, but $3 will contain the comma character before the second field. This setting of $3 will be the proper contents with which to enter GETFLD to do the saved encoding of the second field. For IOD processing, MIOD encodes the second field (exit address) of the IOD instruction.

Then MIOD increments the Zinfld counter in the unit by only one because Zinfld tells Pass 2 how many encoded fields there are in the unit. However, MIOD increments the error message field counter once after the first field has been skipped over and once after GETFLD has encoded the second field. Thus this field counter will locate the proper part of the IOD if an error is detected in GETFLD or in MIOD.

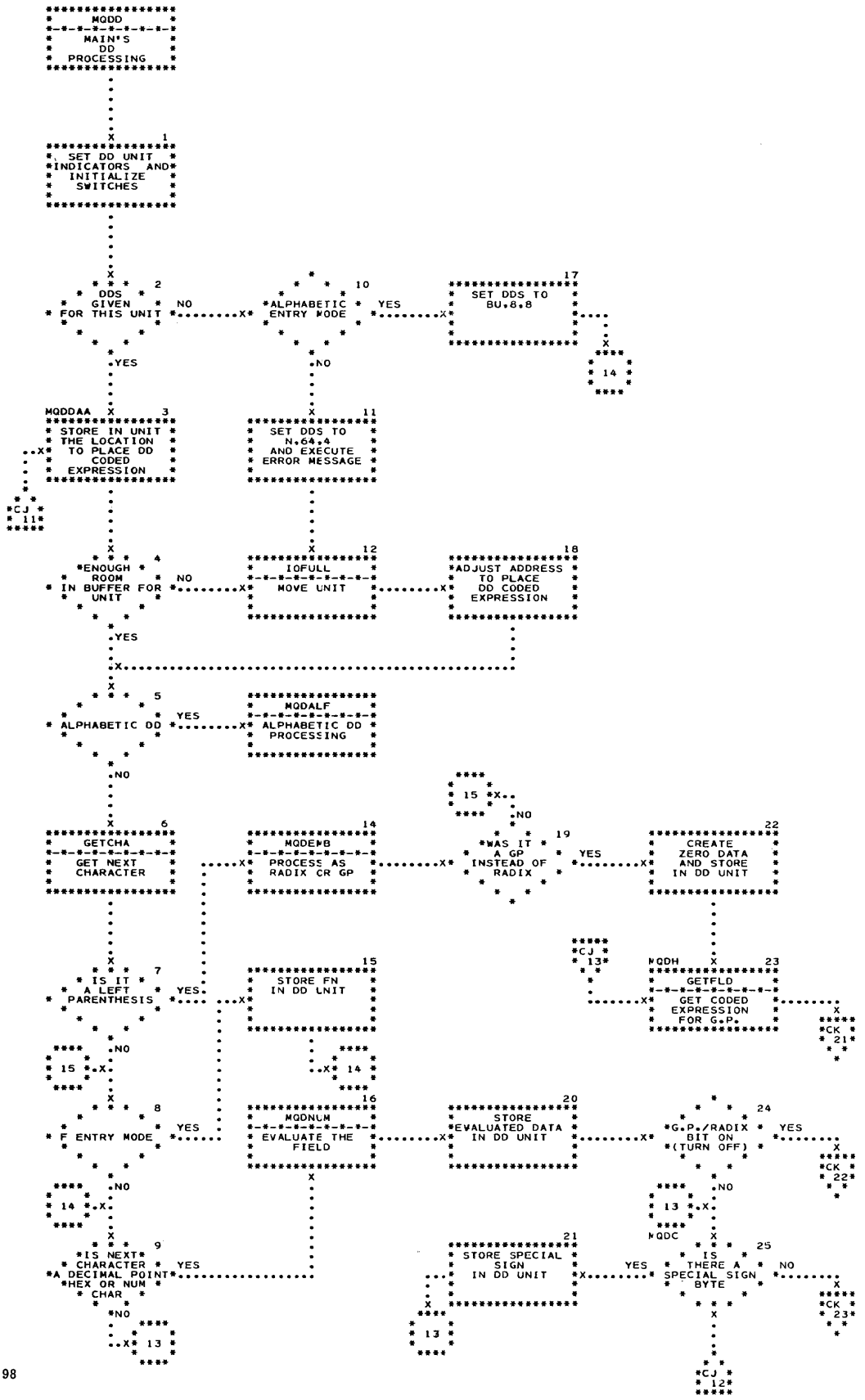
MIOD also sets up the symbol table entry in the Msyte area as MAIN would have done. The reference number counter, initially zero, is incremented and saved in the unused (for IOD) Zilpms field of the unit for later Pass 2 use. The Zifp2 indicator in the unit is turned off so the NMCP subroutine and not DECODE will get control of the processing of the unit in Pass 2. To prevent DECODE from being asked to work on the unit during the main flow of Pass 2, MIOD turns on the Zifall indicator on each IOD or on each REEL unit. In this way, NMCP which has control of the processing of MCP units in Pass 2, will get these expressions evaluated by VALUE directly from NMCP.

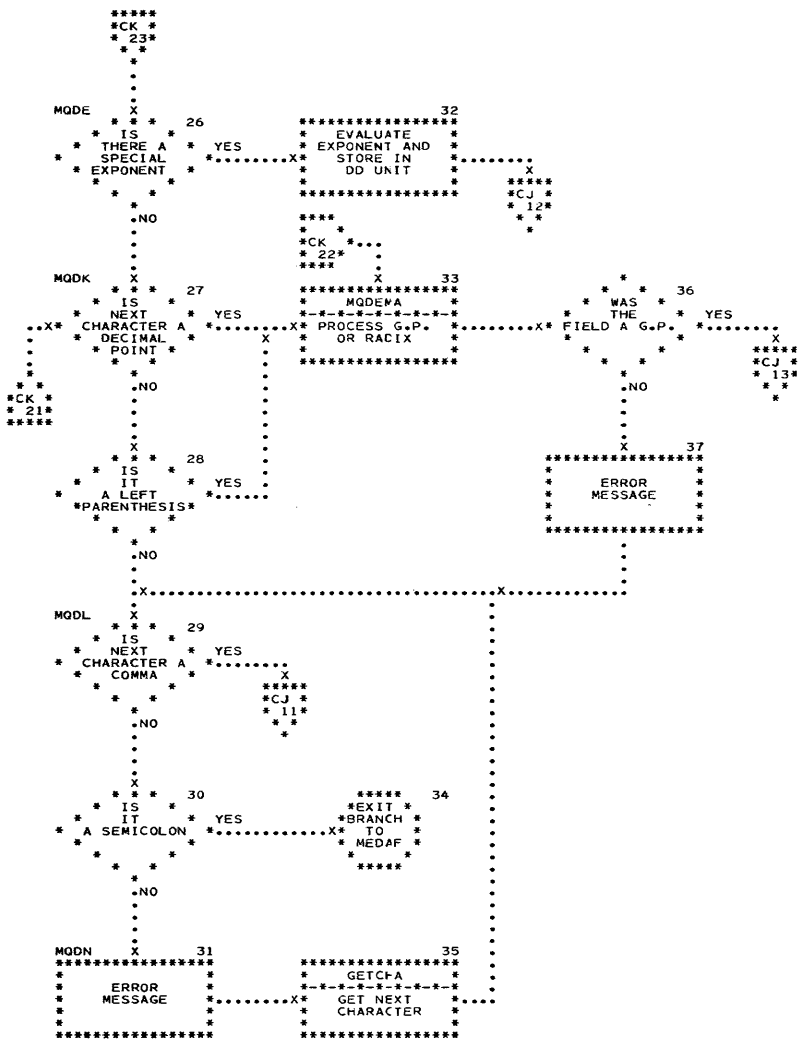
### Boxes 16-19

For a REEL instruction, MIOD does not have much to do. The Zifall indicator is turned on to tell Pass 2 that there are no values to get.

If a name had been specified with a REEL instruction of course it must be set up in the symbol table because ANEXT in Pass 2 must have a name to get from the symbol table for each unit marked as having a name. The symbol table entry will be set up as a 'phoney' or 'special' entry and UITER will ignore the entry during Pass 2.

Then MIOD returns to the location, Medaf, to collect the remaining characters to finish processing the unit in Pass 1.


MQDD
MAIN'S DD PROCESSING
SET DD UNIT INDICATORS AND INITIALIZE SWITCHES
DDS GIVEN FOR THIS UNIT
ALPHABETIC ENTRY MODE
SET DDS TO BU,8,8
MQDDAA
STORE IN UNIT THE LOCATION TO PLACE DD CODED EXPRESSION
SET DDS TO N,64,4 AND EXECUTE ERROR MESSAGE
ENOUGH ROOM IN BUFFER FOR UNIT
IOFULL MOVE UNIT
ADJUST ADDRESS TO PLACE DD CODED EXPRESSION
ALPHABETIC DD
MQDALF ALPHABETIC DD PROCESSING
GETCHA GET NEXT CHARACTER
MQDEMB PROCESS AS RADIX CR GP
WAS IT A GP INSTEAD OF RADIX
CREATE ZERO DATA AND STORE IN DD UNIT
IS IT A LEFT PARENTHESIS
STORE FN IN DD UNIT
MQDH
GETFLD GET CODED EXPRESSION FOR G.P.
MQDNUM EVALUATE THE FIELD
STORE EVALUATED DATA IN DD UNIT
G.P./RADIX BIT ON (TURN OFF)
MQDC
IS NEXT CHARACTER A DECIMAL POINT HEX OR NUM CHAR
STORE SPECIAL SIGN IN DD UNIT
IS THERE A SPECIAL SIGN BYTE
*CJ 11*
*CJ 12*
*CJ 13*
*CK 21*
*CK 22*
*CK 23*
*14*


CK 23
MQDE
IS THERE A SPECIAL EXPONENT
26
YES
NO
EVALUATE EXPONENT AND STORE IN DD UNIT
32
CJ 12
CK 22
MQDK
IS NEXT CHARACTER A DECIMAL POINT
27
YES
NO
MQDEMA PROCESS G.P. OR RACIX
33
CJ 13
WAS THE FIELD A G.P.
36
YES
NO
CK 21
IS IT A LEFT PARENTHESIS
28
YES
ERROR MESSAGE
37
MQDL
IS NEXT CHARACTER A COMMA
29
YES
CJ 11
IS IT A SEMICOLON
30
YES
EXIT BRANCH TO MEDAF
34
MQDN
ERROR MESSAGE
31
GETCHA GET NEXT CHARACTER
35

LINKAGE

MQDD does not have a calling sequence linkage. After main Pass 1 had determined that it is processing a DD unit and has completed the data description, it branches directly to MQDD, which used MQDNUM and DNUM as subroutines. MQDD's final exit branches back to main Pass 1 at Medaf where the final details of processing the unit are accomplished.

DETAILS

MQDD sets up the variable portion of the unit for a numeric DD in the format specified in Appendix A.

FLOW CHART DESCRIPTION

Boxes 2, 10, 17, 11

If no dds is given by the programmer, the dds will be set to BU for alphabetic DD's and N for others.

Box 3

The location in the control word Ibuffw is stored in the DD unit at relative location Ziotpt. 32. This will be the location at which the coded expression of the DD is placed.

Boxes 4, 12, 18

If there is not enough room in the intermediate buffer to contain the entire unit, IOFULL is used to move the previously set up portion of the DD unit to the beginning of the buffer, and the control word addresses are adjusted accordingly.

Box 5

If the unit is an alphabetic DD, control is immediately transferred to MQDALF.

Boxes 6, 7, 14, 19, 22

If there was a left parenthesis, it is followed by either a radix or GP. This number is evaluated by Mqdemb, using CPLTNM, and stored accordingly in the unit. If it was a GP, MQDD assumes that there was no data, and stores a zero coded expression in the DD unit.

Box 8

If there was a Fn-entry mode, the bit Bemv was turned on by main Pass 1.

Boxes 9, 16, 20

The DD numeric data is evaluated by MQDNUM and stored in the DD unit.

Boxes 13, 14

If there is a special sign byte, evaluate and store in the DD entry.

Boxes 26, 32

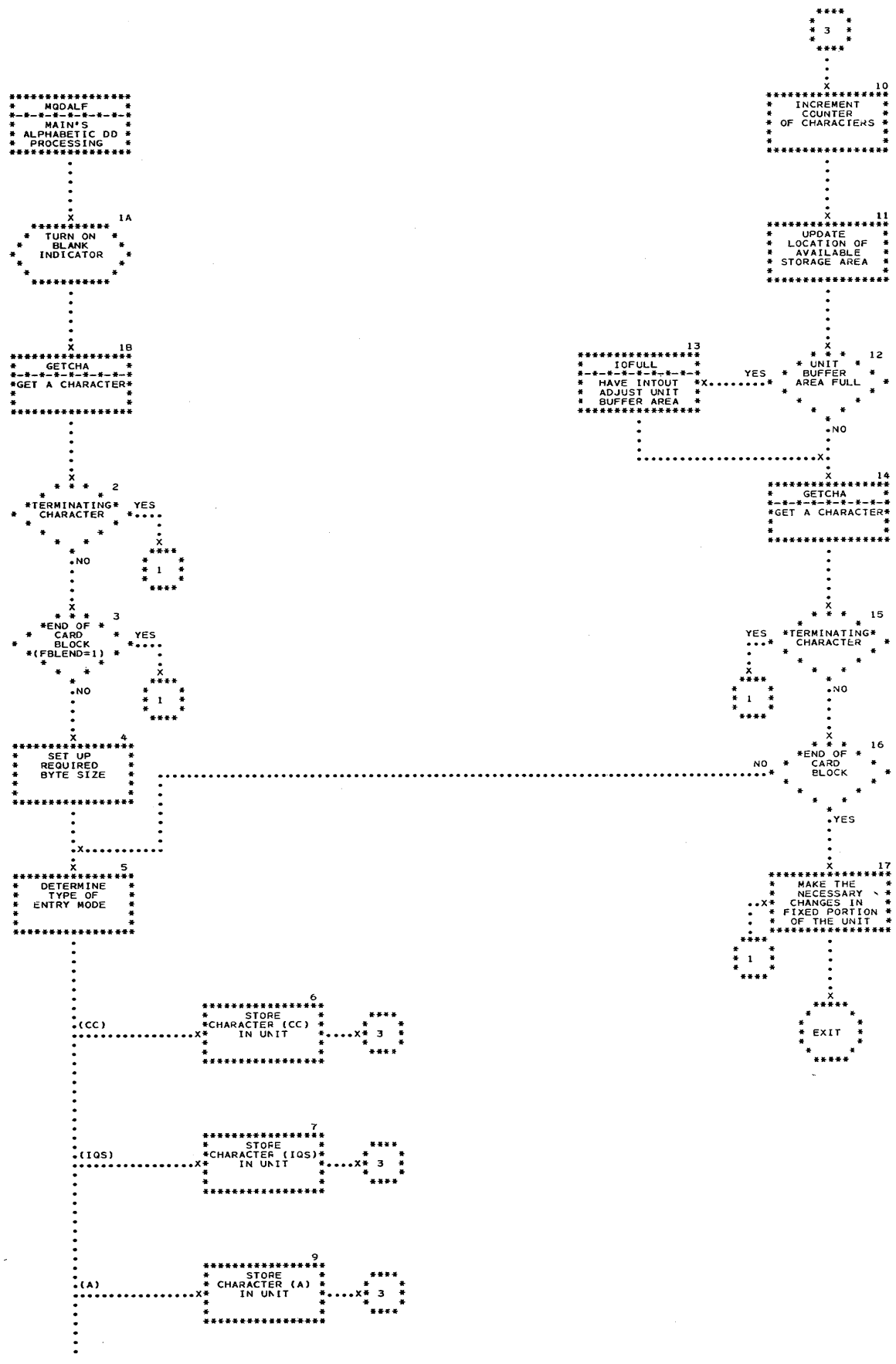
Special exponents are evaluated by MQDNUM and stored in the unit.

Boxes 27, 33, 36, 28, 29

The remaining characters are analyzed. Additional GP's, exponents, or sign bytes if any, return to the proper section of MQDD; another radix at this point would be an error, and is error flagged.

Boxes 30, 34

The final exit from MQDD returns to Medaf in main Pass 1.


MODALF
MAIN'S ALPHABETIC DD PROCESSING
1A
TURN ON BLANK INDICATOR
1B
GETCHA
GET A CHARACTER
2
TERMINATING CHARACTER
YES
NO
3
END OF CARD BLOCK (FBLEND=1)
YES
NO
4
SET UP REQUIRED BYTE SIZE
5
DETERMINE TYPE OF ENTRY MODE
(CC)
6
STORE CHARACTER (CC) IN UNIT
(IQS)
7
STORE CHARACTER (IQS) IN UNIT
(A)
9
STORE CHARACTER (A) IN UNIT
3
10
INCREMENT COUNTER OF CHARACTERS
11
UPDATE LOCATION OF AVAILABLE STORAGE AREA
12
UNIT BUFFER AREA FULL
YES
NO
13
IOFULL
HAVE INTOUT
ADJUST UNIT BUFFER AREA
14
GETCHA
GET A CHARACTER
15
TERMINATING CHARACTER
YES
NO
16
END OF CARD BLOCK
NO
YES
17
MAKE THE NECESSARY CHANGES IN FIXED PORTION OF THE UNIT
1
EXIT

## LINKAGE

### On Entry

1. Main flow branches to MQDALF after MAIN gets from GETCHA the comma immediately preceeding the D field. The comma is still in the value field of $3 when MAIN goes to MQDALF.
2. The value field of the index word at Bem + 1.0 contains the terminating character of the alphabetic DD.
3. The indicators, Bemt, Bemu, and Bemv are set to the appropriate alphabetic entry mode.
4. The value field of the index word, Ibuffw, contains the lower bound of the area where the material is to be put in the expanded instruction unit.
5. The value field of the index word, Ibuflw, contains the address of the upper bound of the available space in the unit.

### On Exit

1. The necessary statement is in the DD unit. (Ibuffw will have been updated accordingly. Ibuflw may or may not have been updated.)
2. The count field at relative location Zinfld in the expanded instruction unit of the DD contains the number of characters in the D field.
3. If the byte size is symbolic, bits 0-23 of the Ziotpt field in the unit contains the number of characters also.
   Bit 24 of Ziotpt = 0 in this case.
   However, if the byte size of the DD is absolute, bits 0-23 of Ziotpt contains the number of bits in the D field.
   Bit 24 of Ziotpt = 1 under this condition.
4. Bit Ziotpt.26 for this unit will be 0 if the entry mode is A, P, or IQS; 1 if the entry mode is CC. (The above fields and indicators are of interest to OUTPUT in Pass 2b.)
5. Bit Ziotpt.27 = 1 if OUTPUT has to perform a possible truncation of the characters in the DD message.
6. After completing the conversion, MQDALF returns to the main flow of Pass 1 with the value field of $3 containing the next non-blank character after the terminating character of the DD.

## USER

MQDD transfers to MQDALF for the processing of the statement field of a DD pseudo-op.

## FLOW CHART DESCRIPTION

### Boxes 1A, 1B

MQDALF has GETCHA get the first character of the alphabetic D field since MQDD comes to MQDALF with the comma separator in the value field of $3. MQDALF must turn on the Gcaz indicator so GETCHA will 'kick out' blanks since a blank is significant in this field.

### Boxes 2, 3; 15, 16

Each time MQDALF uses GETCHA to get a new input character, MQDALF must make two tests before processing the character as part of the alphabetic DD. Of course, on detection of the terminating character, MQDALF stops collecting alphabetic DD characters in the expanded instruction unit and performs the final housekeeping. (The terminating character is in the Bem area with other DD information. See Phase 1 description.)

In case a terminating character is absent from the D field, the end-of-card-block indication from GETCHA causes MQDALF to terminate the DD.

### Box 4

The information, whether the data description of the op is symbolic or absolute, and if absolute, what the byte size is, is included in the fixed portion of the op's expanded instruction unit. Using this information, MQDALF sets up the appropriate entry mode subprocedure to collect the required number of bits for the specified type entry mode.

If the byte size is symbolic which remains unevaluated during Pass 1, MQDALF saves the maximum 8 bits of each of the converted A or IQS characters, or the 12 bits of each of the converted CC characters. Leading zeros are added if necessary. The OUTPUT routine - in Pass 2b - will truncate each of the collected converted characters in the unit for the final binary output.

If the byte size collected by MAIN was absolute, MQDALF adjusts the appropriate subprocedure to store the required number of bits of each converted character. (However, MQDALF forces a 12-bit byte size if entry mode is CC.)

### Boxes 5-9

According to the Bem setting of the Bem indicators, MQDALF transfers to the correct subprocedure and

stores the converted characters in the variable portion of the expanded instruction unit.

Box 10

MQDALF increments the character counter, the contents of which will be later transferred to a field in the fixed-format portion of this DD's expanded instruction unit.

Box 11

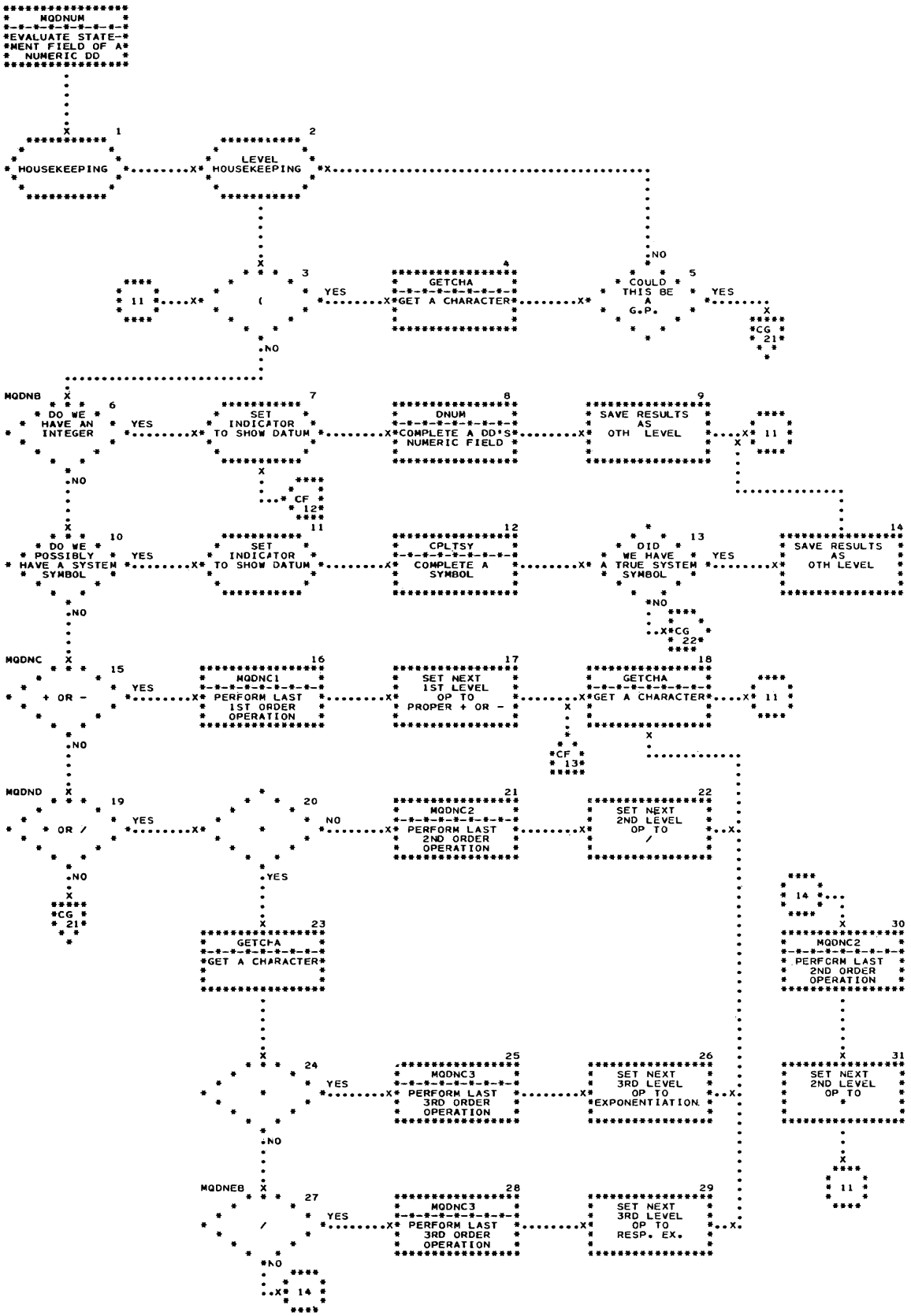
The index word containing the location of available area in the unit is updated.

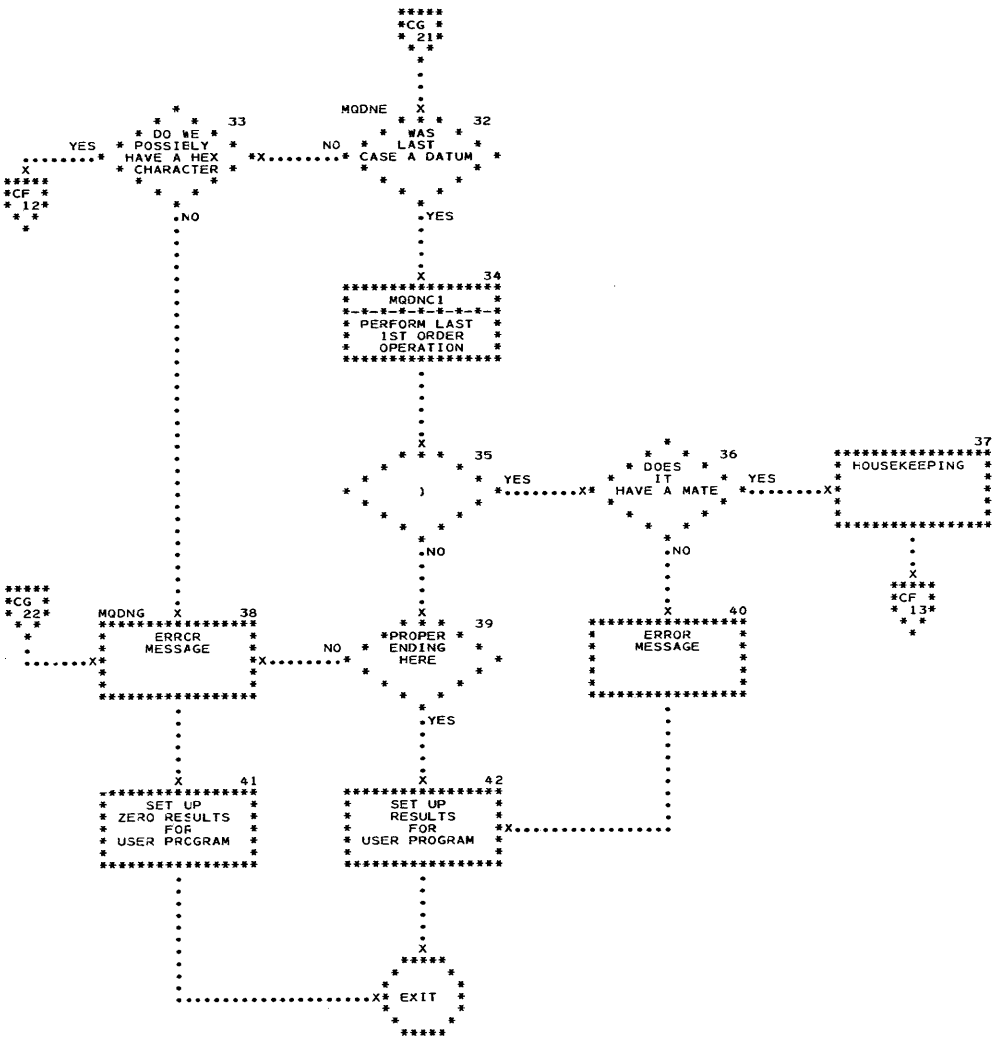
Boxes 12, 13

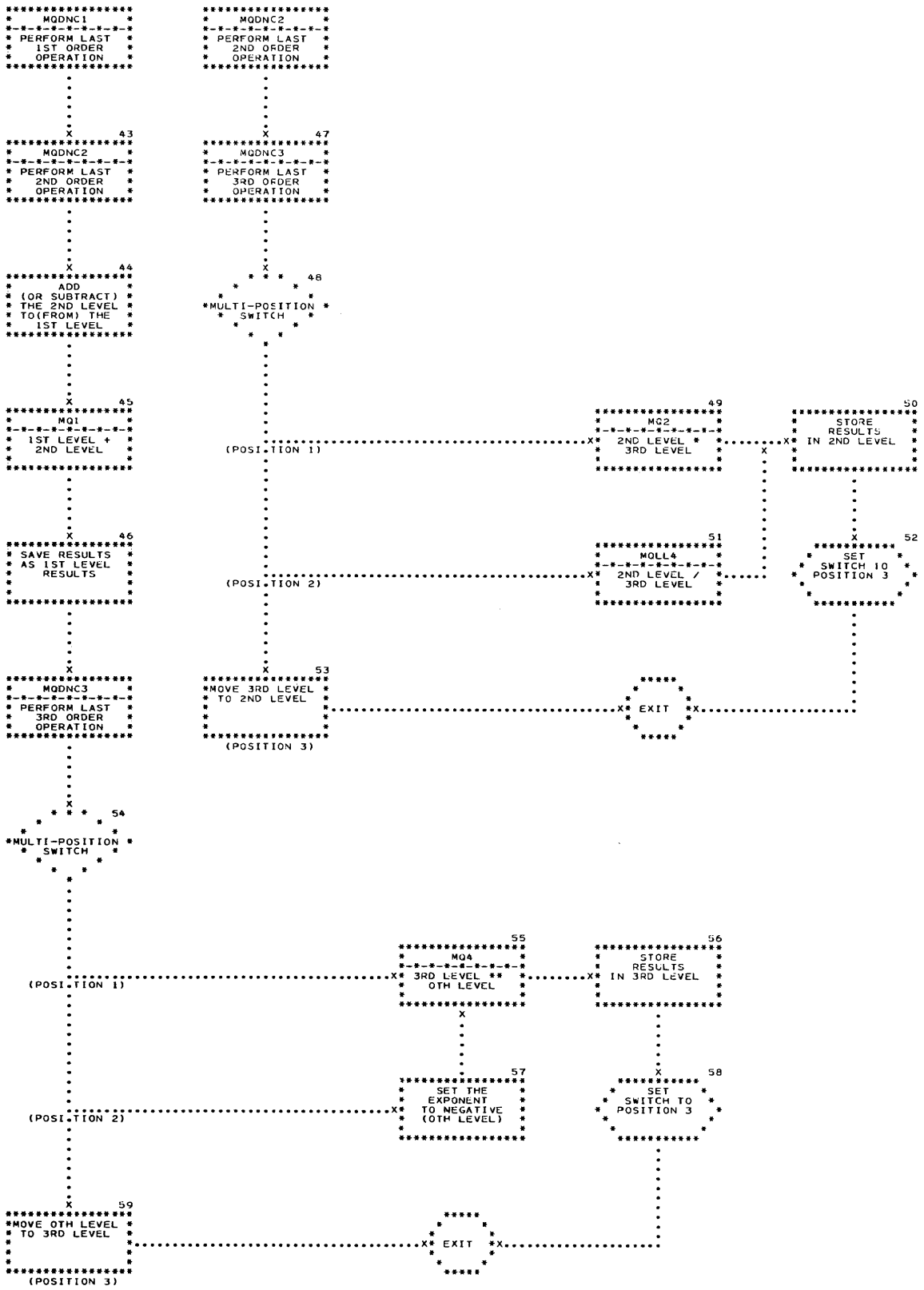
If the usual area allowed for the variable portion of the expanded instruction unit has been exceeded, the buffer for the unit has to be adjusted to make room for the next unit.

Box 17

After the terminating character has been reached, MQDALF adjusts the expanded instruction unit as indicated in the linkage descriptions.


MQDNUM
EVALUATE STATEMENT FIELD OF A NUMERIC DD
1
HOUSEKEEPING
2
LEVEL HOUSEKEEPING
3
(
YES
NO
4
GETCHA
GET A CHARACTER
5
COULD THIS BE A G.P.
YES
NO
CG 21
11
MQDNB
6
DO WE HAVE AN INTEGER
YES
NO
7
SET INDICATOR TO SHOW DATUM
CF 12
8
DNUM
COMPLETE A DD'S NUMERIC FIELD
9
SAVE RESULTS AS OTH LEVEL
11
10
DO WE POSSIBLY HAVE A SYSTEM SYMBOL
YES
NO
11
SET INDICATOR TO SHOW DATUM
12
CPLTSY
COMPLETE A SYMBOL
13
DID WE HAVE A TRUE SYSTEM SYMBOL
YES
NO
CG 22
14
SAVE RESULTS AS OTH LEVEL
MQDNC
15
+ OR -
YES
NO
16
MQDNC1
PERFORM LAST 1ST ORDER OPERATION
17
SET NEXT 1ST LEVEL OP TO PROPER + OR -
18
GETCHA
GET A CHARACTER
11
CF 13
MQDND
19
* OR /
YES
NO
20
21
MQDNC2
PERFORM LAST 2ND ORDER OPERATION
22
SET NEXT 2ND LEVEL OP TO /
CG 21
23
GETCHA
GET A CHARACTER
14
30
MQDNC2
PERFORM LAST 2ND ORDER OPERATION
24
YES
NO
25
MQDNC3
PERFORM LAST 3RD ORDER OPERATION
26
SET NEXT 3RD LEVEL OP TO EXPONENTIATION
31
SET NEXT 2ND LEVEL OP TO
11
MQDNE8
27
/
YES
NO
28
MQDNC3
PERFORM LAST 3RD ORDER OPERATION
29
SET NEXT 3RD LEVEL OP TO RESP. EX.
14


*CG *
* 21*
MQDNE
32
WAS LAST CASE A DATUM
YES
NO
33
DO WE POSSIBLY HAVE A HEX CHARACTER
*CF *
* 12*
NO
YES
34
MQDNC1
PERFORM LAST 1ST ORDER OPERATION
35
)
YES
36
DOES IT HAVE A MATE
YES
37
HOUSEKEEPING
NO
NO
*CG *
* 22*
MQDNG
38
ERRCR MESSAGE
NO
39
PROPER ENDING HERE
40
ERROR MESSAGE
YES
41
SET UP ZERO RESULTS FOR USER PROGRAM
42
SET UP RESULTS FOR USER PROGRAM
EXIT
*CF *
* 13*


MQDNC1
PERFORM LAST 1ST ORDER OPERATION
MQDNC2
PERFORM LAST 2ND ORDER OPERATION
MQDNC2
PERFORM LAST 2ND ORDER OPERATION
MQDNC3
PERFORM LAST 3RD ORDER OPERATION
ADD (OR SUBTRACT) THE 2ND LEVEL TO(FROM) THE 1ST LEVEL
MULTI-POSITION SWITCH
MQ1
1ST LEVEL + 2ND LEVEL
(POSITION 1)
MQ2
2ND LEVEL * 3RD LEVEL
STORE RESULTS IN 2ND LEVEL
SAVE RESULTS AS 1ST LEVEL RESULTS
MQLLA
2ND LEVEL / 3RD LEVEL
SET SWITCH TO POSITION 3
(POSITION 2)
MQDNC3
PERFORM LAST 3RD ORDER OPERATION
MOVE 3RD LEVEL TO 2ND LEVEL
(POSITION 3)
EXIT
MULTI-POSITION SWITCH
(POSITION 1)
MQ4
3RD LEVEL ** OTH LEVEL
STORE RESULTS IN 3RD LEVEL
(POSITION 2)
SET THE EXPONENT TO NEGATIVE (OTH LEVEL)
SET SWITCH TO POSITION 3
MOVE OTH LEVEL TO 3RD LEVEL
EXIT
(POSITION 3)

LINKAGE

On Entry

1. The calling sequence is: SIC, Mqdnuz; B, MQDNUM.
2. The first character of the field is in the value field of $3.

On Exit

1. The non-blank character following the field is in the value field of $3.
2. The double precision result is in the accumulator.

FLOW CHART DESCRIPTION

Boxes 6-14

For each level in the arithmetic operations MQDNUM makes an eight word entry in its push down list at Mqnnd. In this entry two words are reserved for the result from each of the first, second, and third order operations. The remaining two words are for miscellaneous indicators including those designating the type of arithmetic combination between these intermediate result areas in the entry. MQDNUM maintains a common storage area for the result of the fourth order, i.e., each new operand from DNUM.

The current arithmetic operations which may be specified on a numeric DD are:

| First order operations | Second order operations | Third order operations |
|---|---|---|
| +(addition) | *(multiplication) | **(powering) |
| -(subtraction) | /(division) | |

Note that in powering the exponent must be an integer in the range $2^{-18} < n < 2^{+18}$.

While operating within a level, as each new datum is encountered, MQDNUM has it made into a double precision floating point word by DNUM, and then MQDNUM enters the result into the fourth order result area.

Box 13

Remember that CPLTSY sets up the location of the system symbol mathematical constant as a negative quantity in the accumulator.

Boxes 45, 49, 51, 55

These are double precision arithmetic subroutines whose specifications will be described shortly. These double precision arithmetic subroutines as well as DNUM give the using program the results in the accumulator.

Box 33

A radix may be 2-16.

Box 15 ff.

While operating within a level, as each new operator is encountered, MQDNUM combines the intermediate results of an entry in the following manner:

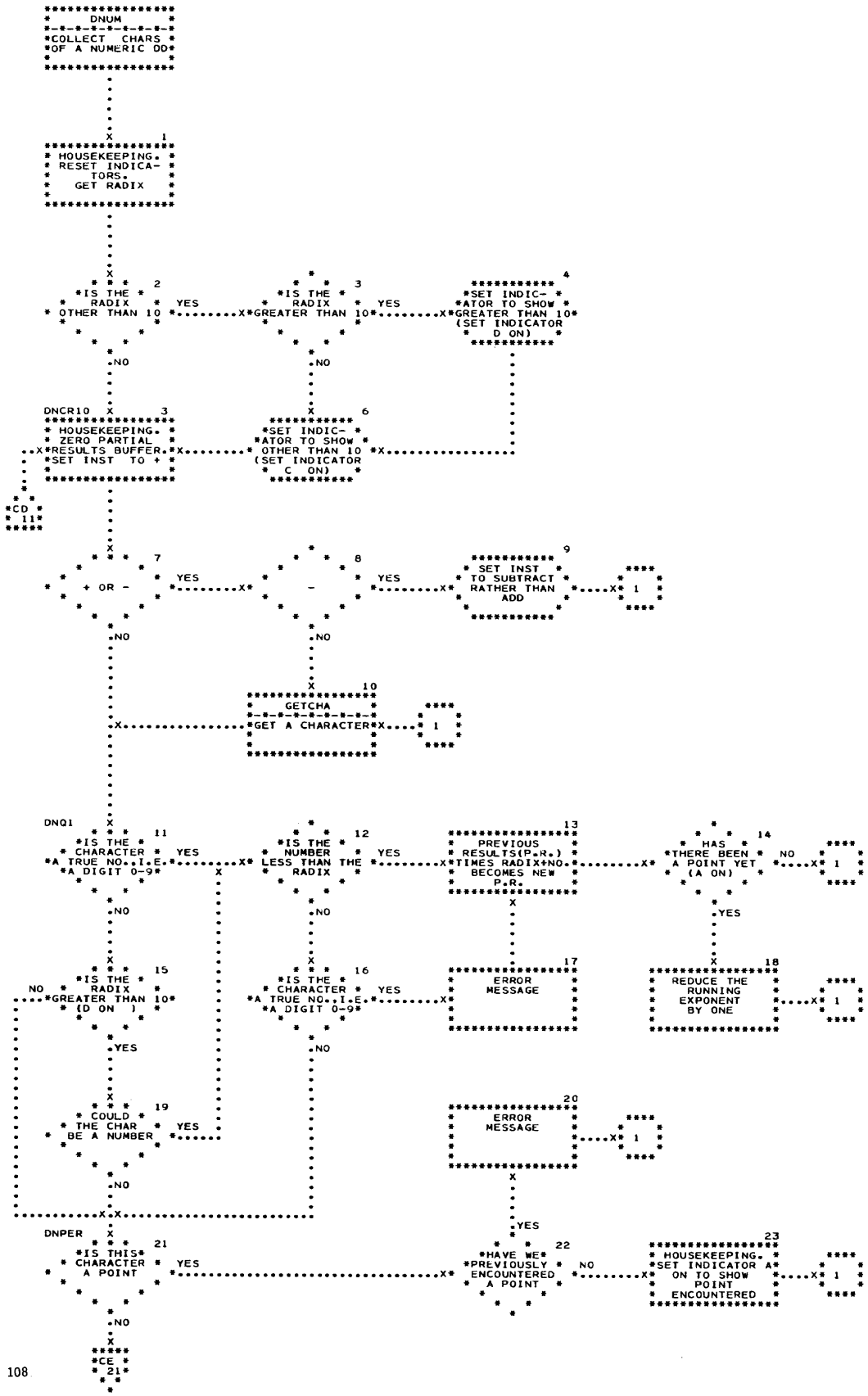
1. For any operator, MQDNUM combines the results of the fourth order with that of the third order according to the corresponding operator indicators; MQDNUM then enters this result in the third order result area and resets the operator indicators of the fourth and the third orders to the transfer setting. (Initially all the operator indicators are in the transfer setting.)
2. For a second order operation (*, /) MQDNUM also combines the result of the third order with that of the second order according to the corresponding indicators; MQDNUM then enters this result in the second order result area and resets the operator indicators of the third and second orders to the transfer setting.
3. For a first order operator (+, -) MQDNUM further combines the result of the second order with that of the first order, etc..
4. After combining the appropriate results in a Mqdnnd entry, MQDNUM adjusts the arithmetic indicators in the entry to the operation just encountered.

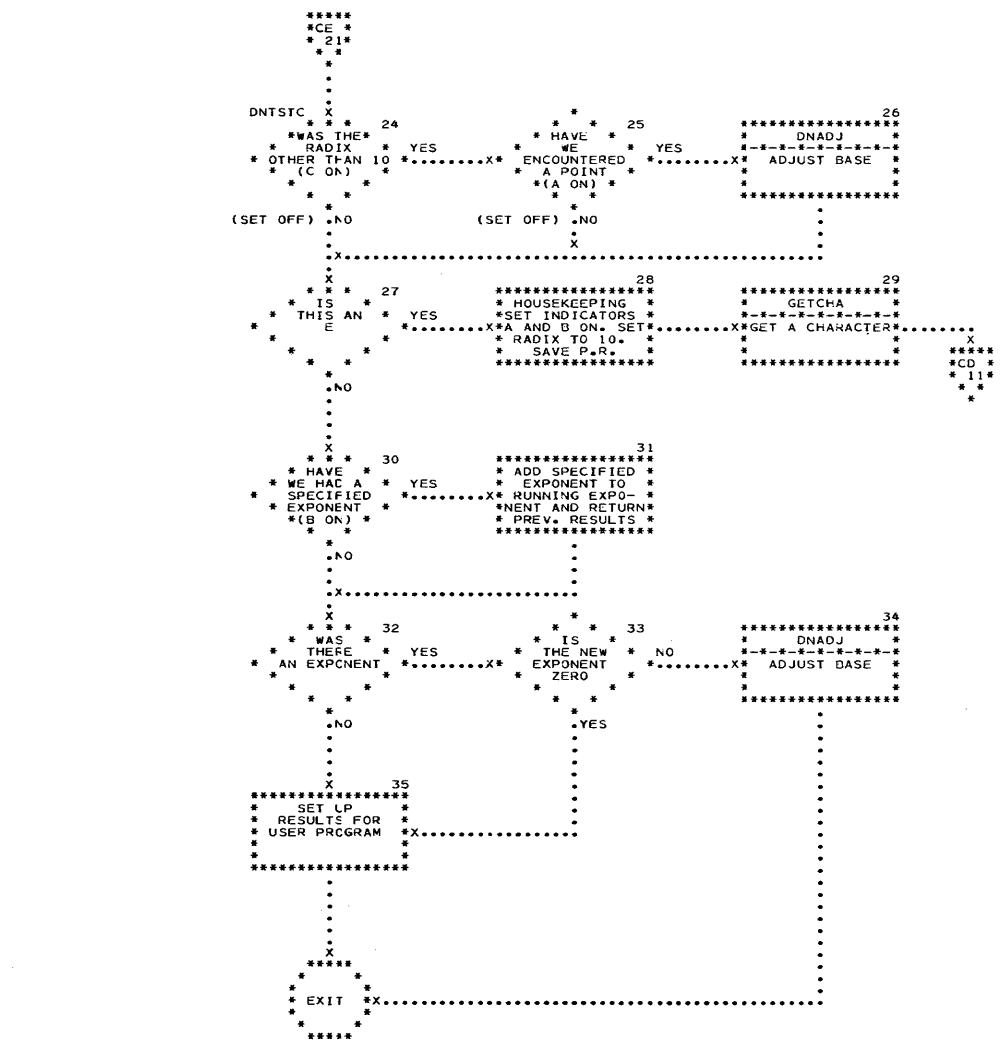
Boxes 32-42

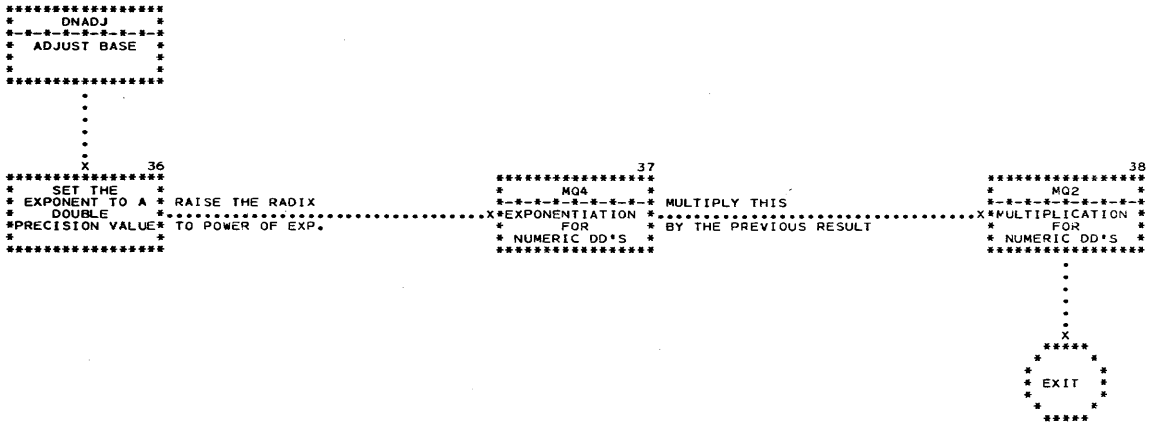
This routine evaluates the field to the terminal character (, , @, or ;) or to the special sign or exponent (S or X).

Box 37

As each non-initial level is completed, the result becomes the contents of the fourth order area in the previous level.


DNUM
COLLECT CHARS OF A NUMERIC DD
1
HOUSEKEEPING. RESET INDICA- TORS. GET RADIX
2
IS THE RADIX OTHER THAN 10
YES
NO
3
IS THE RADIX GREATER THAN 10
YES
NO
4
SET INDIC- ATOR TO SHOW GREATER THAN 10 (SET INDICATOR D ON)
DNCR10
3
HOUSEKEEPING. ZERO PARTIAL RESULTS BUFFER. SET INST TO +
6
SET INDIC- ATOR TO SHOW OTHER THAN 10 (SET INDICATOR C ON)
CD 11
7
+ OR -
YES
NO
8
-
YES
NO
9
SET INST TO SUBTRACT RATHER THAN ADD
1
10
GETCHA
GET A CHARACTER
1
DNQ1
11
IS THE CHARACTER A TRUE NO. I.E. A DIGIT 0-9
YES
NO
12
IS THE NUMBER LESS THAN THE RADIX
YES
NO
13
PREVIOUS RESULTS(P.R.) TIMES RADIX+NO. BECOMES NEW P.R.
14
HAS THERE BEEN A POINT YET (A ON)
NO
YES
15
IS THE RADIX GREATER THAN 10 (D ON )
NO
YES
16
IS THE CHARACTER A TRUE NO. I.E. A DIGIT 0-9
YES
NO
17
ERROR MESSAGE
18
REDUCE THE RUNNING EXPONENT BY ONE
19
COULD THE CHAR BE A NUMBER
YES
NO
20
ERROR MESSAGE
1
DNPER
21
IS THIS CHARACTER A POINT
YES
NO
22
HAVE WE PREVIOUSLY ENCOUNTERED A POINT
NO
23
HOUSEKEEPING. SET INDICATOR A ON TO SHOW POINT ENCOUNTERED
1
CE 21


CE 21
DNTSTC
24
WAS THE RADIX OTHER THAN 10 (C ON)
YES
(SET OFF) NO
25
HAVE WE ENCOUNTERED A POINT (A ON)
YES
(SET OFF) NO
26
DNADJ
ADJUST BASE
27
IS THIS AN E
YES
NO
28
HOUSEKEEPING SET INDICATORS
A AND B ON. SET RADIX TO 10. SAVE P.R.
29
GETCHA
GET A CHARACTER
CD 11
30
HAVE WE HAD A SPECIFIED EXPONENT (B ON)
YES
NO
31
ADD SPECIFIED EXPONENT TO RUNNING EXPONENT AND RETURN
PREV. RESULTS
32
WAS THERE AN EXPONENT
YES
NO
33
IS THE NEW EXPONENT ZERO
NO
YES
34
DNADJ
ADJUST BASE
35
SET LP RESULTS FOR USER PROGRAM
EXIT


DNADJ
ADJUST BASE
36
SET THE EXPONENT TO A DOUBLE PRECISION VALUE
RAISE THE RADIX TO POWER OF EXP.
37
MQ4
EXPONENTIATION FOR NUMERIC DD'S
MULTIPLY THIS BY THE PREVIOUS RESULT
38
MQ2
MULTIPLICATION FOR NUMERIC DD'S
EXIT

DNUM

LINKAGE

On Entry

1. The calling sequence is: SIC, Dnuz; B, DNUM
2. The first character of the operand is in the value field of $3.
3. The radix in effect for this field is in location, Drad. (This radix is set up by MQDD before MQDNUM gets control for processing this field.)

On Exit

1. The double precision floating point result is in the accumulator.
2. The next non-blank character after the operand is in the value field of $3.

USER

MQDNUM

FLOW CHART DESCRIPTION

Box 1

Indicators A, B, C, and D are turned off. Note the operand that is to be collected and evaluated is in the format: $\overset{b}{(\pm}\ n,\ \overset{b}{\pm}\ n.n,$ or $\overset{b}{\pm}\ n.nE\pm e)$.

Box 5, 9

The instruction referred to is the plus part of the calculation indicated in box 13.

Box 19

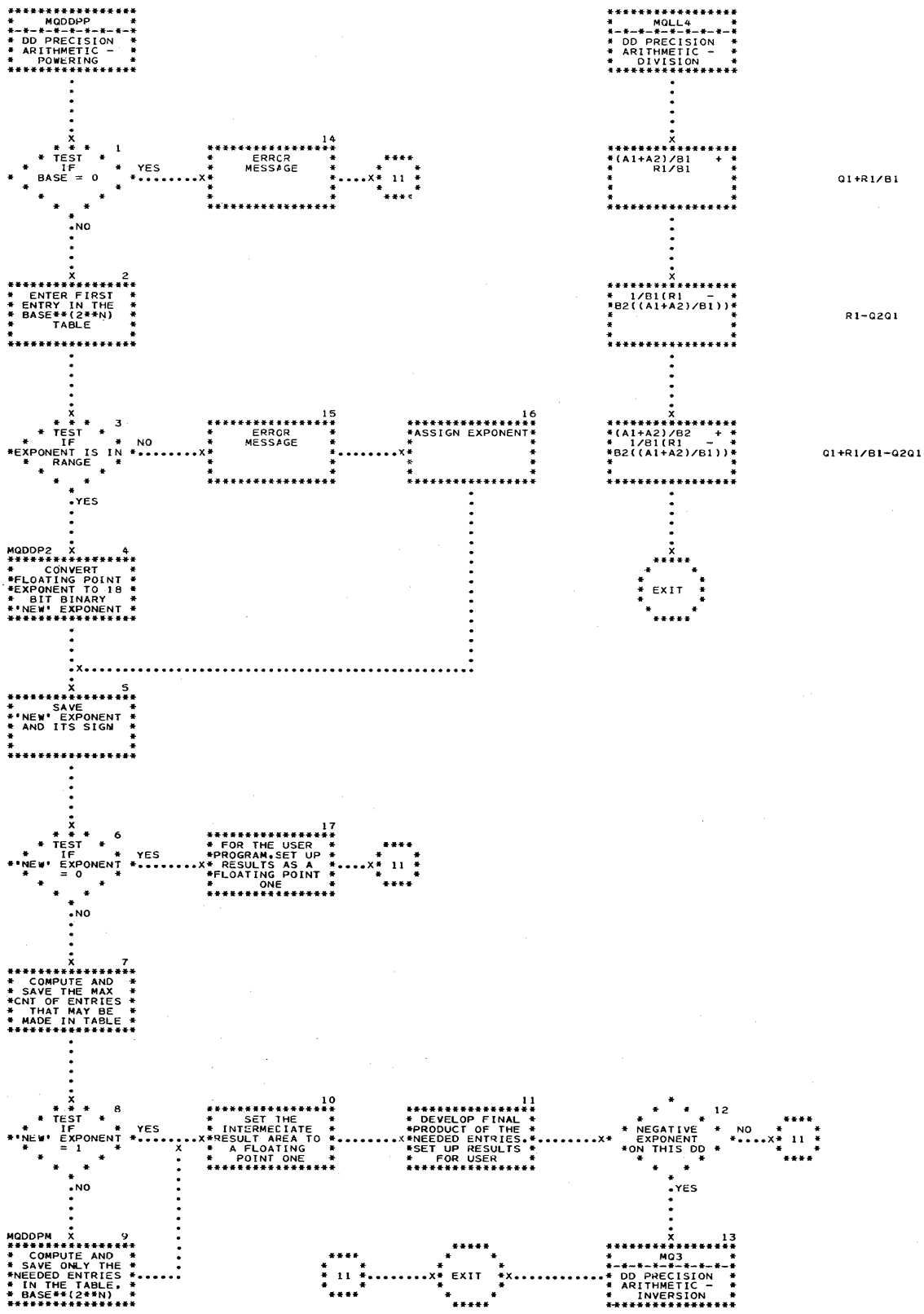
Since a radix can be in the range 2-16, the character can be one of the set A-F.

Boxes 18, 31, 34

Two exponents are mentioned in the flow chart. The exponent which DNUM develops to keep count of the number of places after the point is referred to as a running exponent. The exponent appearing in the source data is referred to as the specified exponent. The previous result is adjusted by each of these exponents to produce the final result.

Box 36

The values to be arithmetically combined by any of the MQDDPA subroutines must be in double precision floating point form.


MQDDPP
DD PRECISION ARITHMETIC - POWERING
TEST IF BASE = 0
YES
NO
ERROR MESSAGE
14
11
ENTER FIRST ENTRY IN THE BASE**(2**N) TABLE
TEST IF EXPONENT IS IN RANGE
NO
YES
ERROR MESSAGE
15
ASSIGN EXPONENT
16
MQDDP2
CONVERT FLOATING POINT EXPONENT TO 18 BIT BINARY 'NEW' EXPONENT
SAVE 'NEW' EXPONENT AND ITS SIGN
5
TEST IF 'NEW' EXPONENT = 0
YES
NO
FOR THE USER PROGRAM, SET UP RESULTS AS A FLOATING POINT ONE
17
11
COMPUTE AND SAVE THE MAX CNT OF ENTRIES THAT MAY BE MADE IN TABLE
7
TEST IF 'NEW' EXPONENT = 1
YES
NO
SET THE INTERMEDIATE RESULT AREA TO A FLOATING POINT ONE
10
DEVELOP FINAL PRODUCT OF THE NEEDED ENTRIES. SET UP RESULTS FOR USER
11
NEGATIVE EXPONENT ON THIS DD
NO
YES
12
MQDDPM
COMPUTE AND SAVE ONLY THE NEEDED ENTRIES IN THE TABLE, BASE**(2**N)
9
11
EXIT
MQLL4
DD PRECISION ARITHMETIC - DIVISION
(A1+A2)/B1 + R1/B1
Q1+R1/B1
1/B1(R1 - B2((A1+A2)/B1))
R1-Q2Q1
(A1+A2)/B2 + 1/B1(R1 - B2((A1+A2)/B1))
Q1+R1/B1-Q2Q1
EXIT
MQ3
DD PRECISION ARITHMETIC - INVERSION
13

MQDDPA

These are subroutines: MQl, MQ2, MQLL4, MQDDPP, MQ3

LINKAGE

On Entry

| | Operation | Calling Sequence | VF of $7 | VF of $8 |
|---|---|---|---|---|
| Addition or subtraction | A ± B | SIC, Mqiz; B, MQl | address of A | address of B |
| Multiplication | A * B | SIC, Mq2z; B, MQ2 | address of A | address of B |
| Division | A / B | SIC, Mqll4z; B, MQLL4 | address of A | address of B |
| Powering | A ** B | SIC, Mqddpw; B, MQDDPP | address of A | address of B |
| Inverting | A **-1 | SIC, Mq3z; B, MQ3 | ---- | address of A |

On Exit

The accumulator contains the double precision floating point results.

RESTRICTION(s)

1. B of A**B must be a whole number in the range, $2^{-18} < B < 2^{+18}$
2. A and B are double precision floating point numbers.

USER(s)

MQDNUM, DNUM

DETAILS

MQl, MQ2, and MQ3 are simple enough not to warrant flow charts. However, the flow chart of the MQDDPP and MQLL4 subroutines are included. (MQ3 is much faster than the MQDDPP subroutine for inverting.)

FLOW CHART DESCRIPTION

Boxes 1, 2, 3

The algorithm for the MQLL4 subroutine is shown in Figure 16.

$$\frac{a_1 + a_2}{b_1 + b_2} = \frac{a_1 + a_2}{b_1}\left(\frac{1}{1 + \frac{b_2}{b_1}}\right)$$

$$= \frac{a_1 + a_2}{b_1}\left(1 - \frac{b_2}{b_1} + \ldots\right) \quad \text{ignore}$$

$$= \left(\frac{a_1 + a_2}{b_1}\right) - \left(\frac{b_2}{b_1}\right)\left(\frac{a_1 + a_2}{b_1}\right)$$

$$= \left(Q_1 + \frac{R_1}{b_1}\right) - \left(Q_2 + \frac{R_2}{b_1}\right)\left(Q_1 + \frac{R_1}{b_1}\right) \quad \text{ignore}$$

$$= Q_1 + \frac{R_1}{b_1} - Q_2 Q_1$$

Figure 16. Algorithm for Division

Boxes 1, 14; 3, 15, 16

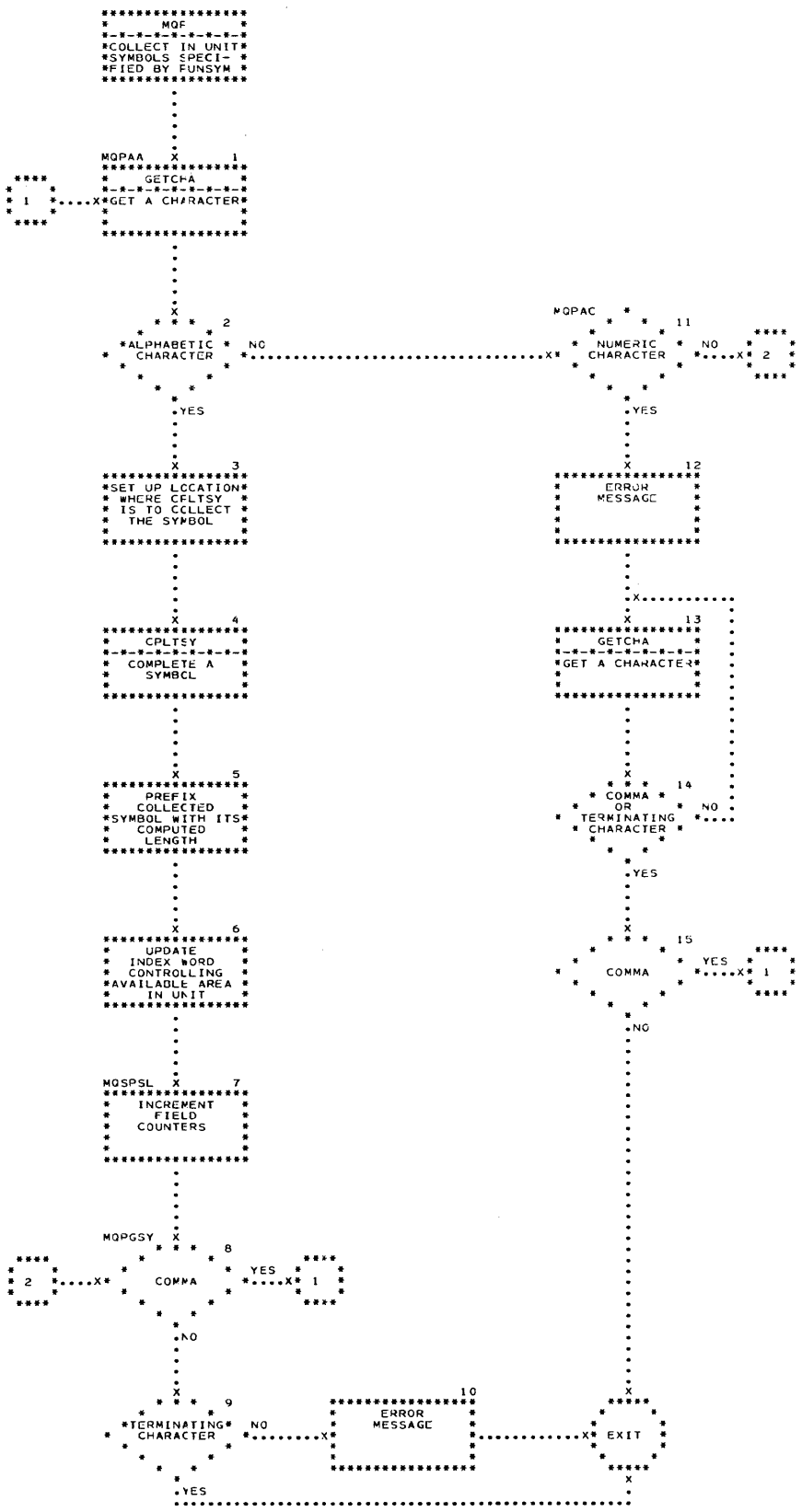
If A in A ** B is zero, MQDDPP gives an error message and puts a floating point zero in the accumulator.

If B in A ** B is not in the specified range, MQDDPP gives an error message and uses instead the appropriate bound in the computation.

When $B < 2^{-18}$, MQDDPP assigns $-1 + 2^{-18}$ to B; when $B < 2^{+18}$, MQDDPP assigns $-1 + 2^{+18}$ to B.

Boxes 2, 4, 5, 7, 8, 9, 10, 11

The powering subroutine adjusts the exponent B so it can fit into an eighteen bit field. (The off/on status of a bit in this 18-bit "new" exponent field determines whether a base $2^{2n}$ type entry is made in the thirty-six word buffer.) Then MQDDPP proceeds to build up the needed powers of the base. Each result is saved as a double precision floating point number in its appropriate slot in the thirty-six word buffer. After the entire "new" exponent has been scanned with the appropriate entries made, then all those entries made will be multiplied together (by now using the "new" exponent as a selector) to produce the final answer.


MQF
COLLECT IN UNIT SYMBOLS SPECI- FIED BY FUNSYM
MQPAA
GETCHA
GET A CHARACTER
ALPHABETIC CHARACTER
NO
YES
SET UP LOCATION WHERE CPLTSY IS TO COLLECT THE SYMBOL
CPLTSY
COMPLETE A SYMBOL
PREFIX COLLECTED SYMBOL WITH ITS COMPUTED LENGTH
UPDATE INDEX WORD CONTROLLING AVAILABLE AREA IN UNIT
MQSPSL
INCREMENT FIELD COUNTERS
MQPGSY
COMMA
TERMINATING CHARACTER
ERROR MESSAGE
EXIT
MQPAC
NUMERIC CHARACTER
ERROR MESSAGE
GETCHA
GET A CHARACTER
COMMA OR TERMINATING CHARACTER
COMMA

LINKAGE

On Entry

MAIN enters MQP through the Pass 1 digit-select, pseudo-op branch table.

On Exit

MQP returns to MAIN where the remaining (non-processed) characters of an instruction are collected.

FLOW CHART DESCRIPTION

Box 1

The character in the value field of $3 is the comma after the op mnemonic since MAIN just came to MQP through the digit-select table. MQP must fetch the first character of each symbol before using CPLTSY to collect the remaining characters of the symbol.

Boxes 2, 11-15

Any field containing a null symbol or a symbol beginning with a number is skipped over by MQP.

Boxes 15, 8, 9

Either of two conditions cause MQP to stop collecting any more characters: first, if the symbol in the statement field of the PUNSYM begins with a non-alphameric character; second, when a terminating character occurs.

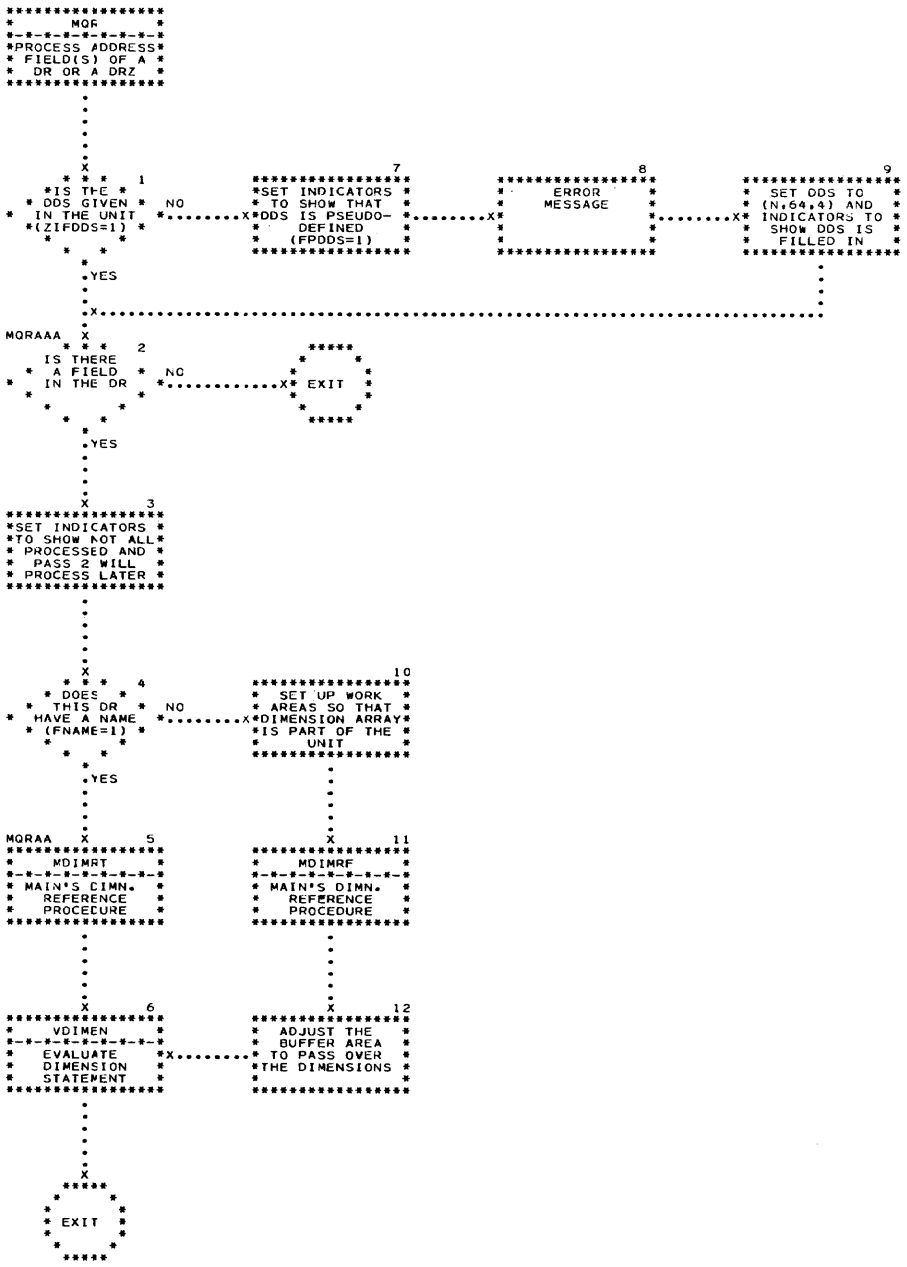
Boxes 3, 4, 5

MQP must set up the index word, Bsymbx, with the location of where the symbol collected by CPLTSY is to be stored. MQP is having the collected symbol put in the variable portion of the PUNSYM's expanded instruction unit. The location given to CPLTSY is set up so that MQP can later prefix the symbol with an eight-bit byte containing the character count of the symbol.

Box 7

Since the PUNSYM instruction will not go through that section in MAIN where the GETFLD subroutine is asked to encode the statement fields, MQP increments the field counters, Zinfld and Field, at this point instead.

STRAP maintains two counters associated with the number of statement fields in the instruction. One counter, the Zinfld field in the fixed portion of each expanded instruction unit, is zeroed during the initialization of each instruction and is incremented every time MAIN detects a new field during Pass 1. Throughout Pass 2 the contents of each Zinfld contains the total number of statement fields in the corresponding instruction. On the other hand the counter Field is a single counter used for all instructions and thus is outside of the expanded instruction unit buffer. Field like each Zinfld field during Pass 1 both is zeroed during the initialization of each instruction and is incremented for each field detected in the statement. However, unlike each Zinfld counter, Field is also zeroed during the initialization of each instruction during Pass 2 also. By the contents of Field, ERROR can determine the number of the field in which an error occurs.


MQR
PROCESS ADDRESS FIELD(S) OF A DR OR A DRZ
1
IS THE DDS GIVEN IN THE UNIT (ZIFDDS=1)
NO
YES
7
SET INDICATORS TO SHOW THAT DDS IS PSEUDO-DEFINED (FPDDS=1)
8
ERROR MESSAGE
9
SET DDS TO (N,64,4) AND INDICATORS TO SHOW DDS IS FILLED IN
MQRAAA
2
IS THERE A FIELD IN THE DR
NO
EXIT
YES
3
SET INDICATORS TO SHOW NOT ALL PROCESSED AND PASS 2 WILL PROCESS LATER
4
DOES THIS DR HAVE A NAME (FNAME=1)
NO
YES
10
SET UP WORK AREAS SO THAT DIMENSION ARRAY IS PART OF THE UNIT
MQRAA
5
MDIMRT
MAIN'S DIMN. REFERENCE PROCEDURE
11
MDIMRF
MAIN'S DIMN. REFERENCE PROCEDURE
6
VDIMEN
EVALUATE DIMENSION STATEMENT
12
ADJUST THE BUFFER AREA TO PASS OVER THE DIMENSIONS
EXIT

LINKAGE

MQR is entered from MAIN through the pseudo-op digit select table.

FLOW CHART DESCRIPTION

Boxes 10-12

When the DR does not have a name, the dimension information is put in the variable portion of the unit. Therefore, the setting of the index words which MAIN used to define the available space in the unit area are used to set up the contents of MDIMRF's index words. MAIN's index word, Ibuffw, is now locating the available area in the variable length portion of the unit where the coded expressions or encoding of any kind is to go. MDIMRF makes up the special dimension configuration to go into the unit (See Appendix A.)

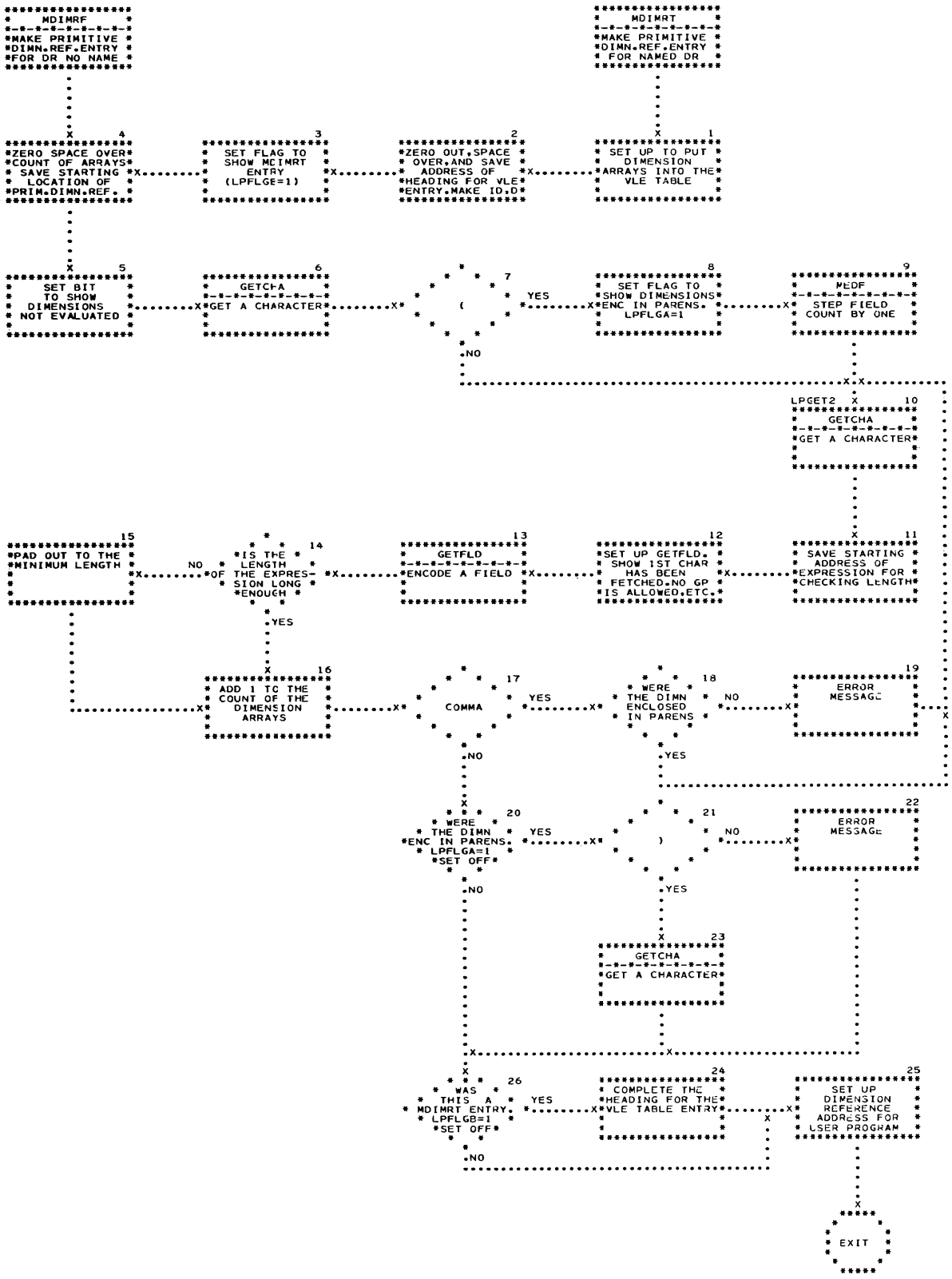
After the primitive dimension information is in the unit, the index words of MAIN must be updated to locate new free area.

Box 6

Before going to VDIMEN, the location of the dimension information is saved in a special field of the unit. Now VDIMEN, a special subroutine of VALUE, tries to evaluate the dimensions. If it can evaluate the dimension information, VDIMEN replaces the previous coded expression with the twenty-four bit products. (See Appendix A.)

Box 5

MQR must enter MDIMRF by the MDIMRT entry because MDIMRT makes the adjustments so that the dimension array can be part of a vle.


MDIMRF
MAKE PRIMITIVE DIMN.REF.ENTRY FOR DR NO NAME
MDIMRT
MAKE PRIMITIVE DIMN.REF.ENTRY FOR NAMED DR
1 SET UP TO PUT DIMENSION ARRAYS INTO THE VLE TABLE
2 ZERO OUT,SPACE OVER,AND SAVE ADDRESS OF HEADING FOR VLE ENTRY,MAKE ID,D
3 SET FLAG TO SHOW MCIMRT ENTRY (LPFLGE=1)
4 ZERO SPACE OVER COUNT OF ARRAYS SAVE STARTING LOCATION OF PRIM.DIMN.REF.
5 SET BIT TO SHOW DIMENSIONS NOT EVALUATED
6 GETCHA GET A CHARACTER
7 (
YES
NO
8 SET FLAG TO SHOW DIMENSIONS ENC IN PARENS. LPFLGA=1
9 MEDF STEP FIELD COUNT BY ONE
LPGET2
10 GETCHA GET A CHARACTER
11 SAVE STARTING ADDRESS OF EXPRESSION FOR CHECKING LENGTH
12 SET UP GETFLD. SHOW 1ST CHAR HAS BEEN FETCHED.NO GP IS ALLOWED,ETC.
13 GETFLD ENCODE A FIELD
14 IS THE LENGTH OF THE EXPRESSION LONG ENOUGH
NO
YES
15 PAD OUT TO THE MINIMUM LENGTH
16 ADD 1 TO THE COUNT OF THE DIMENSION ARRAYS
17 COMMA
YES
NO
18 WERE THE DIMN ENCLOSED IN PARENS
NO
YES
19 ERROR MESSAGE
20 WERE THE DIMN ENC IN PARENS. LPFLGA=1 *SET OFF*
YES
NO
21 )
NO
YES
22 ERROR MESSAGE
23 GETCHA GET A CHARACTER
24 COMPLETE THE HEADING FOR THE VLE TABLE ENTRY
25 SET UP DIMENSION REFERENCE ADDRESS FOR USER PROGRAM
26 WAS THIS A MDIMRT ENTRY. LPFLGB=1 *SET OFF*
YES
NO
EXIT

LINKAGE

On Entry

1. The calling sequence -- SIC, Mdimrz; B, MDIMRF -- is used to put just the dimension information into the available area of the expanded instruction unit located by the index word, Mdimrx. (The dimension data for a DR or a DRZ without a name is put into the unit.)
2. The calling sequence -- SIC, Mdimrz; B, MDIMRT -- is used to make an entire dimension entry in the vle portion of the symbol table. (The dimension data for a DR or DRZ with a name or for a SYN is put in the symbol table.)
3. The value field of $3 contains the character appearing just before the dimension in the statement field.
4. The value field of the index word, Mdimrx, locates the available area in the expanded instruction unit of the DR or the DRZ without a name.

On Exit

1. The value field of $3 will contain the character immediately following the last dimension field.
2. The value field of the index word, Mdimrx, will be updated accordingly.
3. The value field of $1 contains the location (dimension reference address) of the dimension information whether the dimension information is in the vle entry or in the expanded instruction unit.

USERS

MQR, MQS

FLOW CHART DESCRIPTION

Boxes 1, 4

Why must the dimension information be set up in a new vle rather than in the current unit when the dimension is on a SYN or on a named DR instruction? The answer is that the dimension information must always be in memory for these cases since a reference to the name of the SYN or of the DR may occur anywhere in the source program. The entire symbol table remains in memory throughout the assembly; the buffer of the units is not necessarily entirely in memory during any phase of the assembly.

Boxes 1, 2, 3; 26, 24

If the primitive dimension entry is to be set up as part of a vle, the MDIMRF's index words which define its buffer area can be set up inside of the subroutine. MDIMRT get these bounds from the table management control block of the symbol table. Of course, when the dimension entry is part of the vle other information (pertinent to the vle) must also be set up by MDIMRF. The starting position of the vle is saved so later the total length can be computed and inserted in the 13-bit field reserved for this purpose in the vle because the vle by its vary name is not of constant length. Since the vle contains dimension information in the function (i.e., the vle is not in the format of the regular symbol table vle), the (D) character is put in the six-bit ID field. Note a regular symbol table entry (with an entry in the dictionary portion as well as in the vle portion of the symbol table) will be made later at the end of Pass 1 for the name on the DR. The vle entry being built up now does not have a corresponding dictionary portion.

MAIN will take the location of the dimension information supplied by MDIMRF and put this dimension reference address in the unit. Then later when MAIN is setting up the symbol table entry for the name on the unit, MAIN inserts the dimension reference address in the field reserved for it in the function portion of the symbol table entry, Of course, the final housekeeping of MDIMRF must include updating the index words locating the available area in the vle table.

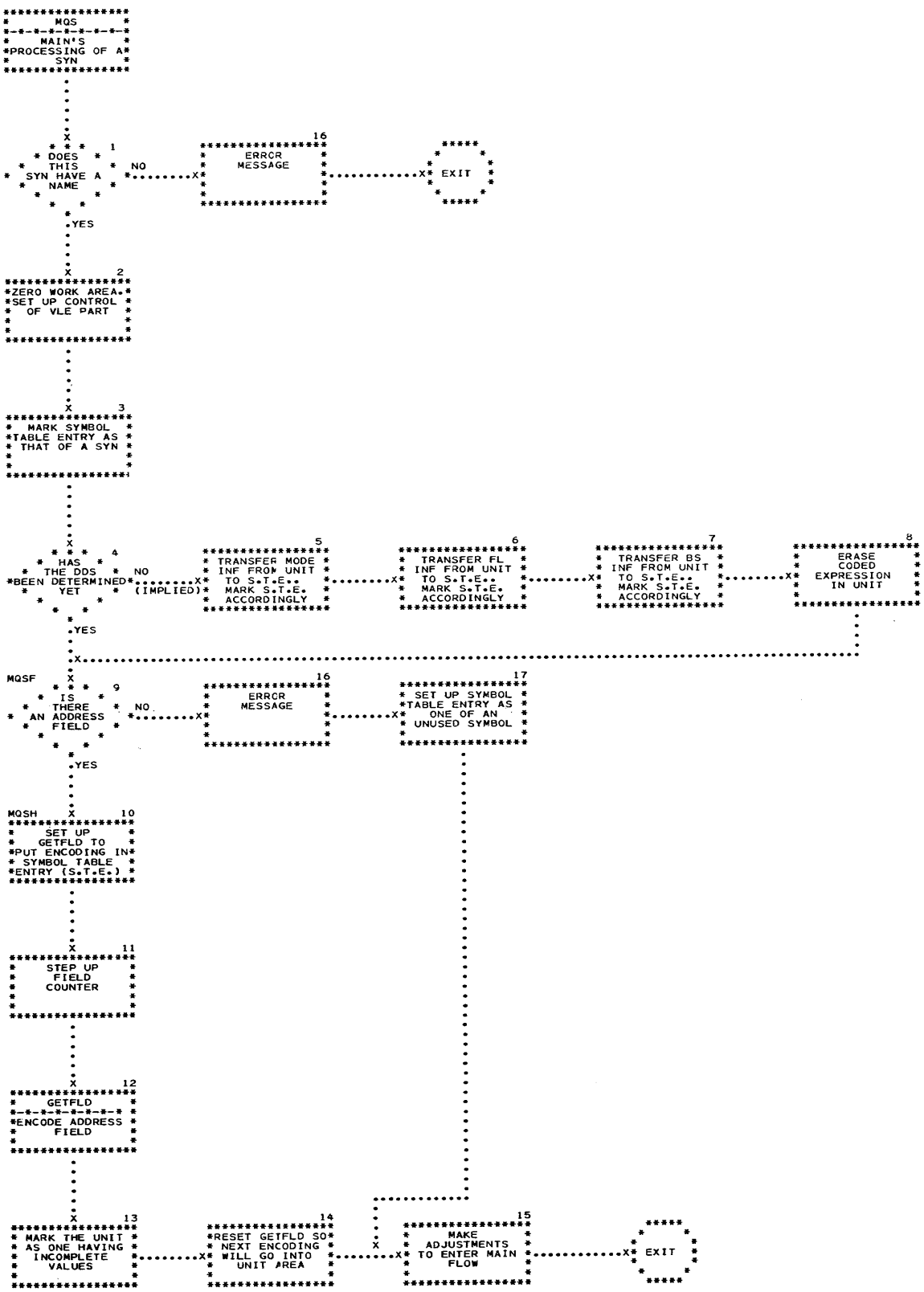
Boxes 4, 5, 16

The five bit count field in the dimension vle is incremented by one after GETFLD encodes each dimension. The if-evaluated indicator in the vle is set to zero to indicate the dimension vle contains the un-evaluated coded expression for each dimension. Later after VDIMEN has evaluated and formed the product of the dimensions, VDIMEN will turn this indicator on.

Boxes 11-15

In the dimension entry a coded expression is made for each of the n dimensions specified. Each coded expression must be at least 24-bits in length because later in the same area previously occupied by the

coded expressions will be the 24-bit evaluated dimension products. (See Appendix A.) The saving of the starting location is done not only for making the minimum length tests but also for marking the prefix to the coded expressions if padding is added to the coded expression.


MQS
MAIN'S PROCESSING OF A SYN
DOES THIS SYN HAVE A NAME
NO
ERROR MESSAGE
EXIT
YES
ZERO WORK AREA. SET UP CONTROL OF VLE PART
MARK SYMBOL TABLE ENTRY AS THAT OF A SYN
HAS THE DDS BEEN DETERMINED YET
NO (IMPLIED)
TRANSFER MODE INF FROM UNIT TO S.T.E.. MARK S.T.E. ACCORDINGLY
TRANSFER FL INF FROM UNIT TO S.T.E.. MARK S.T.E. ACCORDINGLY
TRANSFER BS INF FROM UNIT TO S.T.E.. MARK S.T.E. ACCORDINGLY
ERASE CODED EXPRESSION IN UNIT
MQSF
IS THERE AN ADDRESS FIELD
NO
ERROR MESSAGE
SET UP SYMBOL TABLE ENTRY AS ONE OF AN UNUSED SYMBOL
MQSH
SET UP GETFLD TO PUT ENCODING IN SYMBOL TABLE ENTRY (S.T.E.)
STEP UP FIELD COUNTER
GETFLD
ENCODE ADDRESS FIELD
MARK THE UNIT AS ONE HAVING INCOMPLETE VALUES
RESET GETFLD SO NEXT ENCODING WILL GO INTO UNIT AREA
MAKE ADJUSTMENTS TO ENTER MAIN FLOW
EXIT

LINKAGE

On Entry

MQS is entered from the pseudo-op digit select table with the comma in the value field of $3.

On Exit

MQS returns to the main flow of MAIN with a comment or semicolon in the value field of $3.

FLOW CHART DESCRIPTION

Box 1

If the instruction has a name, Fname will have been turned on by GETCHA.

Boxes 2, 3

MQS, not MAIN, initially sets up Msyte for the name on the SYN instruction. Thus all the work area has to be zeroed before MQS turns on the miscellaneous indicators in the vle.

Boxes 4, 8

If the data description is determined, MAIN will have turned on Zifdds during the processing of the data description in main flow. A reference to the name on the SYN instruction can occur in any unit, but the units are not always in memory while on the other hand the symbol table is. Therefore, the data description information and the address field information must be put in the vle of the symbol table entry. Since the encoding is now in vle, the location of available area in the unit is stepped back.

Boxes 10, 14

Main flow maintains the index word controling available area in the buffer for the GETFLD encoding. But now since the encoding of the address field is to go into the symbol table entry some adjustments have to be made before entering GETFLD. After the encoding is put in the vle, of course these special entrance conditions must be cancelled because besides GETFLD's use of Bcfldx to locate the area where the encoding is to be put,

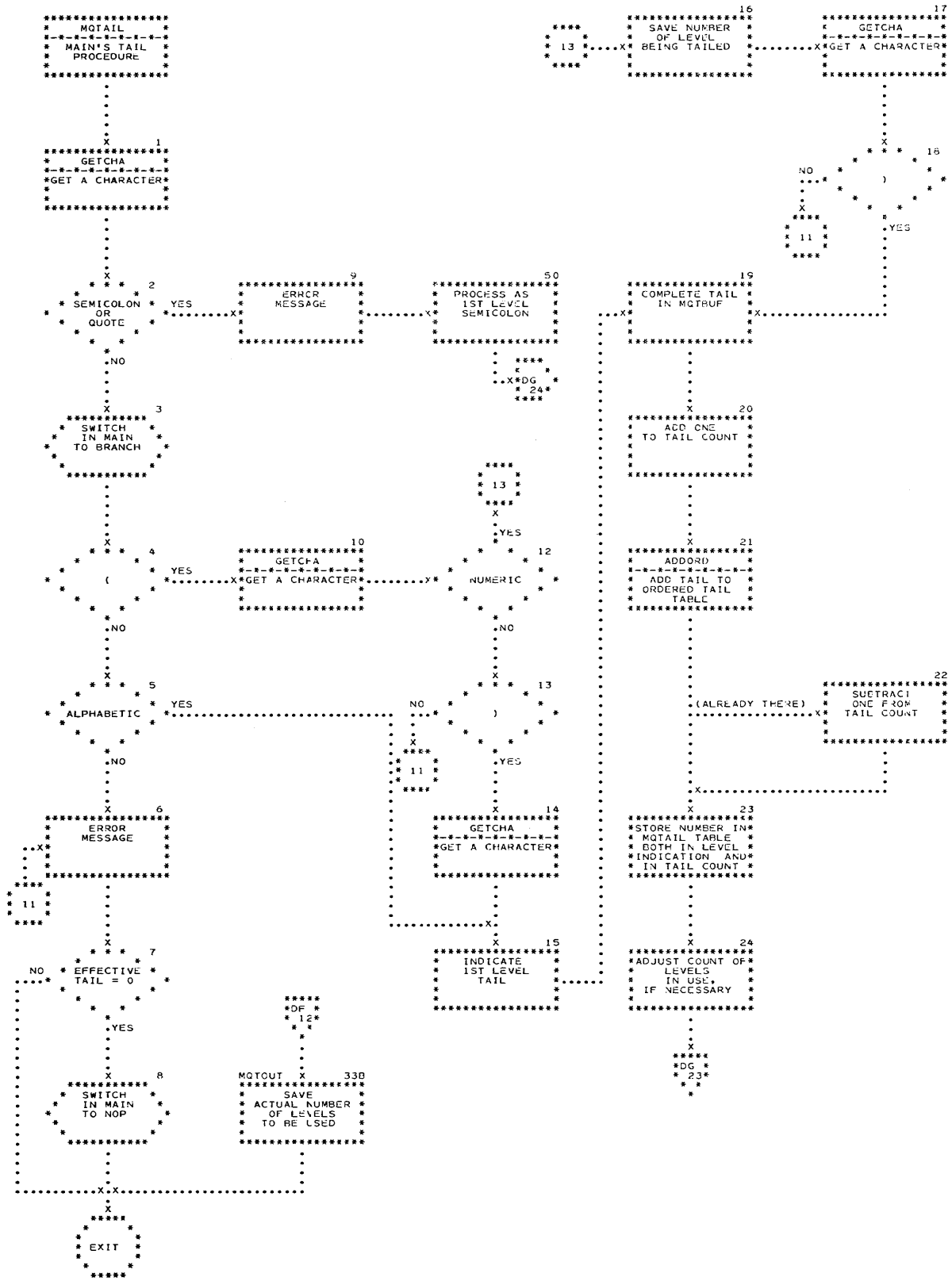
MAIN also uses the updated Bcfldx to determine where the contents of GETCHA's statement buffer is to be inserted in the unit.

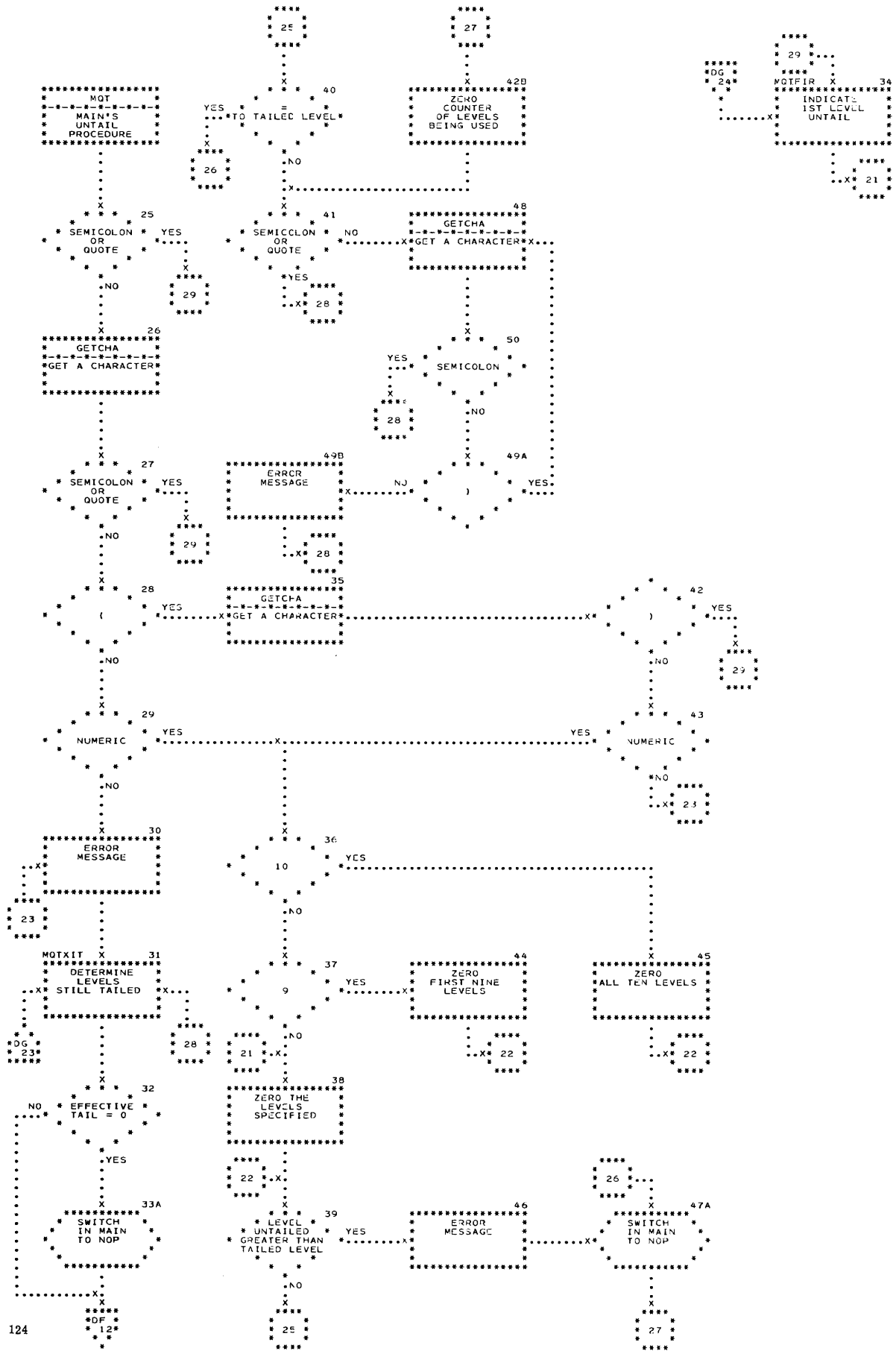
Box 13

Zifall is turned off for Pass 2.

Box 16

If the SYN does not have a name, the remaining characters of the instruction are merely collected so they can be printed on the listing. They do not receive any other processing or analyzation.


MQTAIL
MAIN'S TAIL PROCEDURE
1
GETCHA
GET A CHARACTER
2
SEMICOLON OR QUOTE
YES
NO
9
ERROR MESSAGE
50
PROCESS AS 1ST LEVEL SEMICOLON
DG 24
3
SWITCH IN MAIN TO BRANCH
4
(
YES
NO
10
GETCHA
GET A CHARACTER
12
NUMERIC
YES
NO
5
ALPHABETIC
YES
NO
13
)
YES
NO
11
6
ERROR MESSAGE
14
GETCHA
GET A CHARACTER
7
EFFECTIVE TAIL = 0
NO
YES
15
INDICATE 1ST LEVEL TAIL
DF 12
8
SWITCH IN MAIN TO NOP
MQTOUT
338
SAVE ACTUAL NUMBER OF LEVELS TO BE USED
EXIT
16
SAVE NUMBER OF LEVEL BEING TAILED
13
17
GETCHA
GET A CHARACTER
18
)
NO
YES
11
19
COMPLETE TAIL IN MQTBUF
20
ADD ONE TO TAIL COUNT
21
ADDORD
ADD TAIL TO ORDERED TAIL TABLE
(ALREADY THERE)
22
SUBTRACT ONE FROM TAIL COUNT
23
STORE NUMBER IN MQTAIL TABLE BOTH IN LEVEL INDICATION AND IN TAIL COUNT
24
ADJUST COUNT OF LEVELS IN USE, IF NECESSARY
DG 23


MQT
MAIN'S UNTAIL PROCEDURE
25
SEMICOLON OR QUOTE
26
GETCHA
GET A CHARACTER
27
SEMICOLON OR QUOTE
28
(
29
NUMERIC
30
ERROR MESSAGE
MQTXIT
31
DETERMINE LEVELS STILL TAILED
DG 23
32
EFFECTIVE TAIL = 0
33A
SWITCH IN MAIN TO NOP
DF 12
40
= TO TAILED LEVEL
41
SEMICOLON OR QUOTE
42B
ZERO COUNTER OF LEVELS BEING USED
48
GETCHA
GET A CHARACTER
50
SEMICOLON
49B
ERROR MESSAGE
49A
)
35
GETCHA
GET A CHARACTER
42
)
43
NUMERIC
36
10
37
9
21
38
ZERO THE LEVELS SPECIFIED
39
LEVEL UNTAILED GREATER THAN TAILED LEVEL
44
ZERO FIRST NINE LEVELS
45
ZERO ALL TEN LEVELS
46
ERROR MESSAGE
47A
SWITCH IN MAIN TO NOP
DG 24
MQTFIR
34
INDICATE 1ST LEVEL UNTAIL
YES
NO
25
26
27
28
29
30
31
32
33A
34
21
22
23
25
27
28
29
26

These subroutines are sometimes referred to jointly by TAILOR.

LINKAGE

MQTAIL and MQT are entered from MAIN through the pseudo op digit select table.

DETAILS

MQTAIL subroutine's information primarily consists of: a string of ten consecutive 8-bit bytes each containing the number of the tail associated with one of the ten possible levels, a counter for the number of tails in use, the number of the highest level specified, and the number associated with the tail most recently entered into the table of tails.

The method of internally representing the tails in use is to append the collected symbol with a special tailing configuration of up to eleven A8 characters (*, $n_1$, $n_2$, $n_3$, $n_4$, ... $n_{10}$). The asterisk character, which is an illegal character for a symbol, indicates there is a special tailing configuration. The n represents the number of the tail; the position of the n in the string corresponds to the level of the tail. The string maintained in MQTAIL's table is purposely preceded by an A8 asterisk character, so when MAIN has to indicate tailing on a symbol, MAIN can just pick up directly from the MQTAIL table the appropriate number of fields. However, CPLTSY masks MQTAIL's table onto the CPLTSY tail table which then can be altered, if necessary, as described in CPLTSY.

FLOW CHART DESCRIPTION

Boxes 21 - 24

Each tail is numbered by MQTAIL and saved in an ordered table. This ordered table has a control block other than the one used for the ordered symbol table. The number assigned to the tail in its entry is saved both in the field of the string which corresponds to the current level and in the tail counter. Both are in MQTAIL's table. If a tail has been previously specified, of course, the number associated with the original tail will be used. The counter designating the number of tails in use must also be updated. If the current level is greater than the number of tails in use, the counter of the number of tails in use will be raised. If no tails have been specified at levels in between, the corresponding fields in the string will be zero.

Boxes 31 - 33

After MQTAIL or MQT has processed the TAIL or UNTAIL pseudo op, the string in the MQTAIL table must be scanned. This scan determines two things, first whether or not Pass 1 should continue appending the tailing configuration to symbols collected, and second how many actual levels of tails are to be used if the internal tailing is to continue. It is quite possible that the last pseudo op might have been a (TAIL(6),;) where there were only four levels of tailing prior to this pseudo op occurring. So strictly speaking the count of levels in use is six (from box 24), but the last two tails are blank and really do not need to be indicated in the configuration appended to a symbol. This scan working backwards detects the highest level in use, i.e., the highest rank of a field in the string with a non-zero content. The count of levels in use will then be reset, accordingly.

Boxes 10, 35, 43

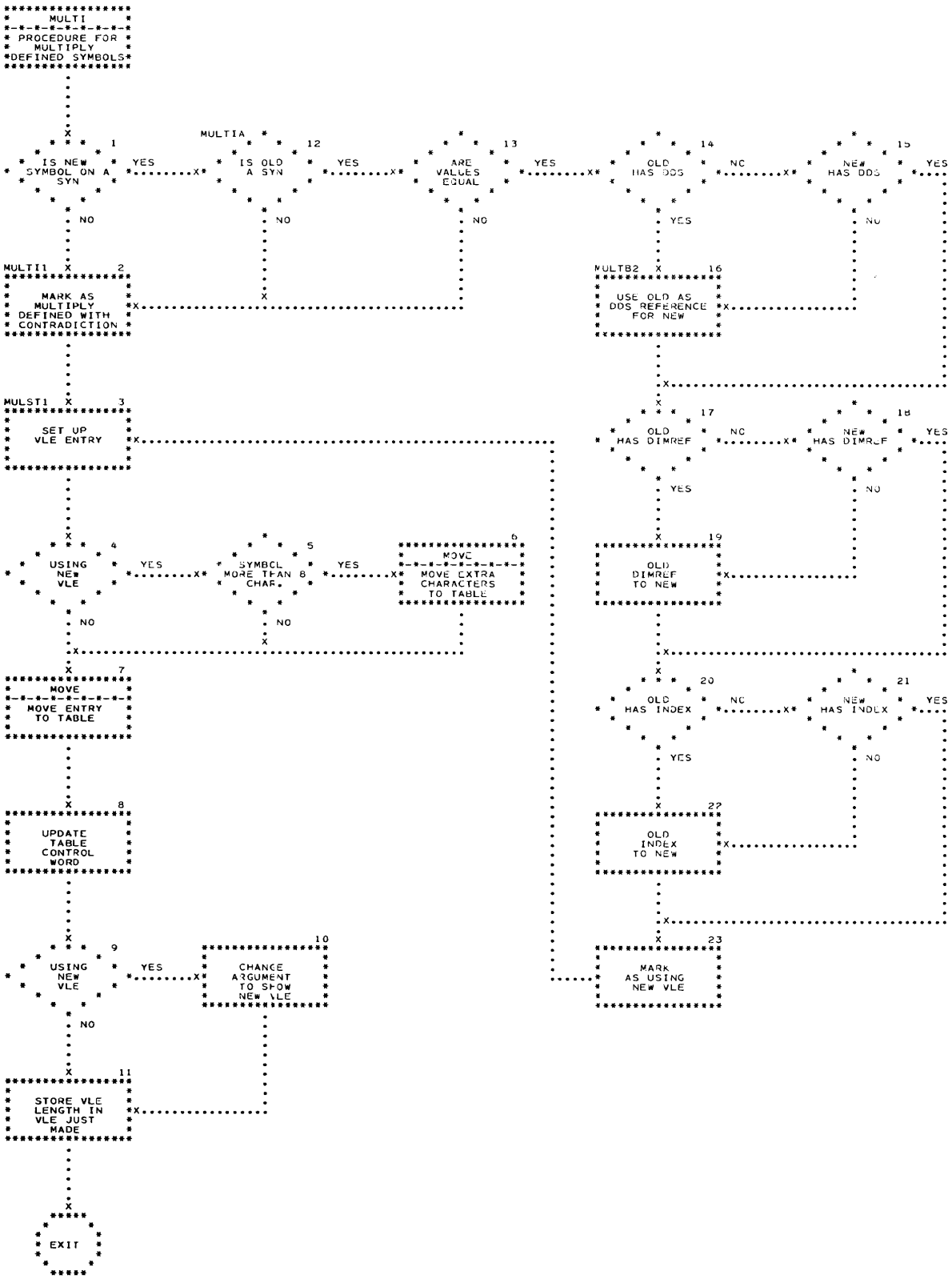
Since GETCHA only gets one character at a time for the using program, GETCHA will be called upon twice to collect the level number whose first digit is one.

Box 38

If a level greater than eight should fall through to this box, only 'n mod(8)' fields in the string will be zeroed.

Boxes 44, 45

The appropriate fields in the string of level indications are zeroed.


MULTI
PROCEDURE FOR MULTIPLY DEFINED SYMBOLS
MULTIA
1
IS NEW SYMBOL ON A SYN
YES
NO
12
IS OLD A SYN
YES
NO
13
ARE VALUES EQUAL
YES
NO
14
OLD HAS DDS
NO
YES
15
NEW HAS DDS
YES
NO
MULTI1
2
MARK AS MULTIPLY DEFINED WITH CONTRADICTION
MULTB2
16
USE OLD AS DDS REFERENCE FOR NEW
MULST1
3
SET UP VLE ENTRY
17
OLD HAS DIMREF
NO
YES
18
NEW HAS DIMREF
YES
NO
4
USING NEW VLE
YES
NO
5
SYMBOL MORE THAN 8 CHAR.
YES
NO
6
MOVE MOVE EXTRA CHARACTERS TO TABLE
19
OLD DIMREF TO NEW
7
MOVE MOVE ENTRY TO TABLE
20
OLD HAS INDEX
NO
YES
21
NEW HAS INDEX
YES
NO
8
UPDATE TABLE CONTROL WORD
22
OLD INDEX TO NEW
9
USING NEW VLE
YES
NO
10
CHANGE ARGUMENT TO SHOW NEW VLE
23
MARK AS USING NEW VLE
11
STORE VLE LENGTH IN VLE JUST MADE
EXIT

## DETAILS

Multiply-defined symbols are detected in Pass 1 and MULTI is the specialized subroutine which treats them. A multiply-defined symbol may be either a legally multiply-defined symbol or an illegally multiply-defined symbol with contradictions.

If the symbol is legally multiply-defined, MULTI will change the dictionary portion of the symbol table entry to locate in the vle portion of the symbol table the new updated vle. This updated vle contains all the accumulated information so far specified on the symbol. Any of the properties given on a previous definition will still hold.

If the symbol is an illegal multiply-defined-with-contradictions symbol, then MULTI will not change the dictionary portion of the symbol table entry. The dictionary portion of the illegal multiply-defined symbol will still locate that vle which contains the old parameters.

In either case, there will be at least a basic vle entry made in the symbol table for each name encountered. (For the subroutine, ANEXT, in Pass 2 there must be a corresponding vle in the symbol table associated with each expanded instruction unit marked as having a name.) MULTI has the additional entry made in the vle table for the multiply-defined symbol.

MULTI also marks a multiply-defined symbol with (or without) contradictions as such in its vle entry. NUNDSY in Pass 2b will then be able to detect the appropriate entries in the symbol table to be printed on the listing.

It is possible to add a data description, dimension properties, and/or index to a quantity previously defined. Consider the following example where the symbol is a legal multiply-defined symbol:

```
A     SYN, 32.0
A     SYN (BU, 32), 32.0 ($2)
```

However, it must be noted that the values for the symbol, i.e., for 'A' in the example, during assembly time are the same.

## FLOW CHART DESCRIPTION

### Boxes 14-22

MAIN has already set up in the Msyte intermediate buffer the function part of the vle portion of the current symbol table entry. Of course, none of the current vle material is in the symbol table because after the subroutine, ADDORD discovered the symbol was multiply-defined, ADDORD effectively went to MULTI. MULTI now makes use of the information already set up in forming the new vle that will go into the vle table. On a multiply-defined symbol without contradictions, it is not possible to overrule a data description, dimension property, and/or index previously specified.

### Boxes 16, 19

The dimension reference (dimref) address field in the function of the vle locates the dimension information usually in another part of the same function. The data description reference (dds ref) address locates the beginning of the function portion of that vle which contains the needed data description information.

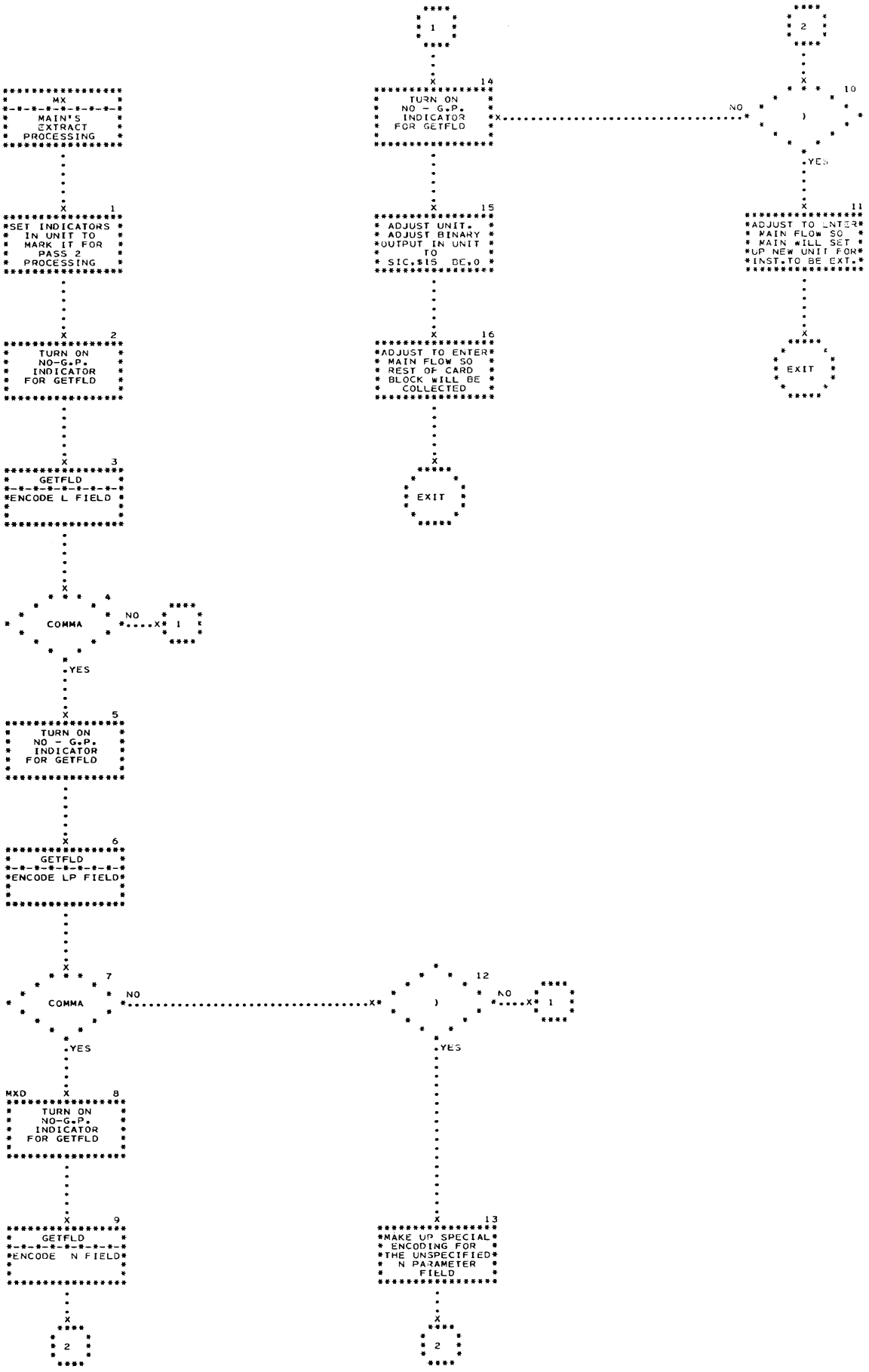
### Box 8

The address of available area in the vle table is updated. The count of the number of symbols remains the same.

The following case illustrates an illegal multiply-defined symbol with contradictions:

```
B     L, JOE
B     L, JOE($2)
```

Here the illegal symbol 'B' has different values during the assembly process. The symbol table entry for the first 'B' is used in all references to this symbol.


MX
MAIN'S EXTRACT PROCESSING
1
SET INDICATORS IN UNIT TO MARK IT FOR PASS 2 PROCESSING
2
TURN ON NO-G.P. INDICATOR FOR GETFLD
3
GETFLD
ENCODE L FIELD
4
COMMA
NO
1
YES
5
TURN ON NO - G.P. INDICATOR FOR GETFLD
6
GETFLD
ENCODE LP FIELD
7
COMMA
NO
YES
MXD
8
TURN ON NO-G.P. INDICATOR FOR GETFLD
9
GETFLD
ENCODE N FIELD
2
1
TURN ON NO - G.P. INDICATOR FOR GETFLD
14
NO
ADJUST UNIT. ADJUST BINARY OUTPUT IN UNIT TO SIC,$IS DC,0
15
ADJUST TO ENTER MAIN FLOW SO REST OF CARD BLOCK WILL BE COLLECTED
16
EXIT
12
)
NO
1
YES
MAKE UP SPECIAL ENCODING FOR THE UNSPECIFIED N PARAMETER FIELD
13
2
2
10
)
NO
YES
11
ADJUST TO ENTER MAIN FLOW SO MAIN WILL SET UP NEW UNIT FOR INST. TO BE EXT.
EXIT

LINKAGE

On Entry

MX is entered from the pseudo-op digit select table with the left parenthesis still in the value field of $3.

On Exit

MX returns to the main flow of MAIN with the right parenthesis in the value field of $3.

FLOW CHART DESCRIPTION

Box 1

The Zifp2 indicator is turned on to show that there are coded expressions in this unit to be decoded by DECODE; Zifall indicator is turned off to indicate that not all the values of this unit are obtained.

Boxes 2, 5, 8

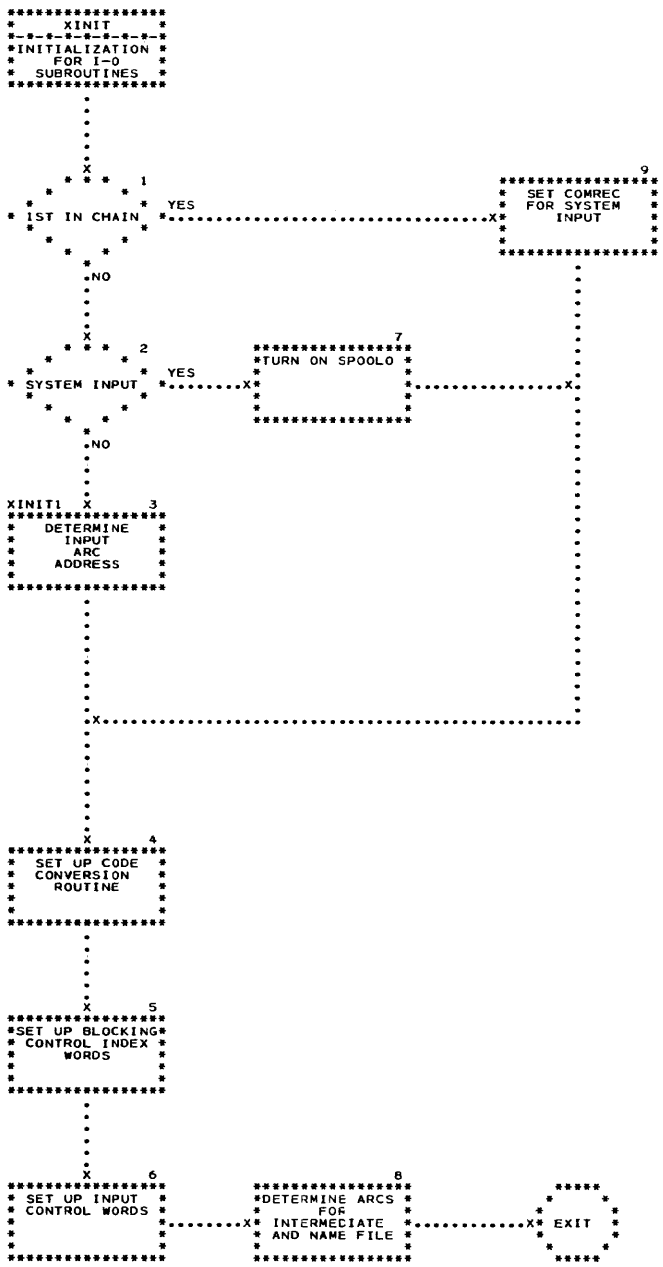
To prevent a general parenthetical field integer entry from being allowed on the parameter, an indicator is turned on before entry into the encoding subroutine, GETFLD.

Boxes 3, 6, 9, 11

For the EXT statement Pass 1 creates 2 units. The first unit is for the parameters, i.e., the GETFLD coded expressions for the parameters are put in the variable part of the first unit. The second unit is for the binary output producing instruction on which the EXT op is to operate. Note the statement to be included in the first unit is considered complete after the right parenthesis has been received from GETCHA. The statement to be included in the second unit is considered complete when the usual terminating semicolon is received from GETCHA. When the second unit is being processed of course it will be treated in Pass 1 as if it were a regular unit. Pass 2 will evaluate the expression and save their values. The OUTPUT routine performs the necessary extracting on the binary output of the second unit.

Box 16

The only processing of the remaining characters will be to collect them in the unit for later listing purposes.


XINIT
INITIALIZATION FOR I-O SUBROUTINES
1
1ST IN CHAIN
YES
NO
9
SET COMREC FOR SYSTEM INPUT
2
SYSTEM INPUT
YES
NO
7
TURN ON SPOOLO
XINIT1
3
DETERMINE INPUT ARC ADDRESS
4
SET UP CODE CONVERSION ROUTINE
5
SET UP BLOCKING CONTROL INDEX WORDS
6
SET UP INPUT CONTROL WORDS
8
DETERMINE ARCS FOR INTERMEDIATE AND NAME FILE
EXIT

LINKAGE

On Entry

The calling sequence is: SIC, Xret; B, XINIT.

On Exit

The following is accomplished before return is made:

1. The input source is determined and proper MCP calling sequences are set up.
2. The code of the input data is determined and linkage to convert routines is set-up.
3. The blocking control is set up according to the number of cards per arc of input.
4. The input buffer is filled.

FLOW CHART DESCRIPTION

Box 1

The COMREC is examined to determine if STRAP II is first in the chain.

Box 9

The eighth word in COMREC is set by STRAP II to indicate System Input, card code, and a blocking factor of one. Spool0 is turned on.

Boxes 2, 7

If System Input is indicated by the COMREC, Spool0 is turned on.

Box 3

The input arc address is obtained from bits 0-17 of word 8 of the COMREC and saved in word 0 of the input buffer. The value field of the blocking control is initialized to word 1 of the input buffer.

Box 4

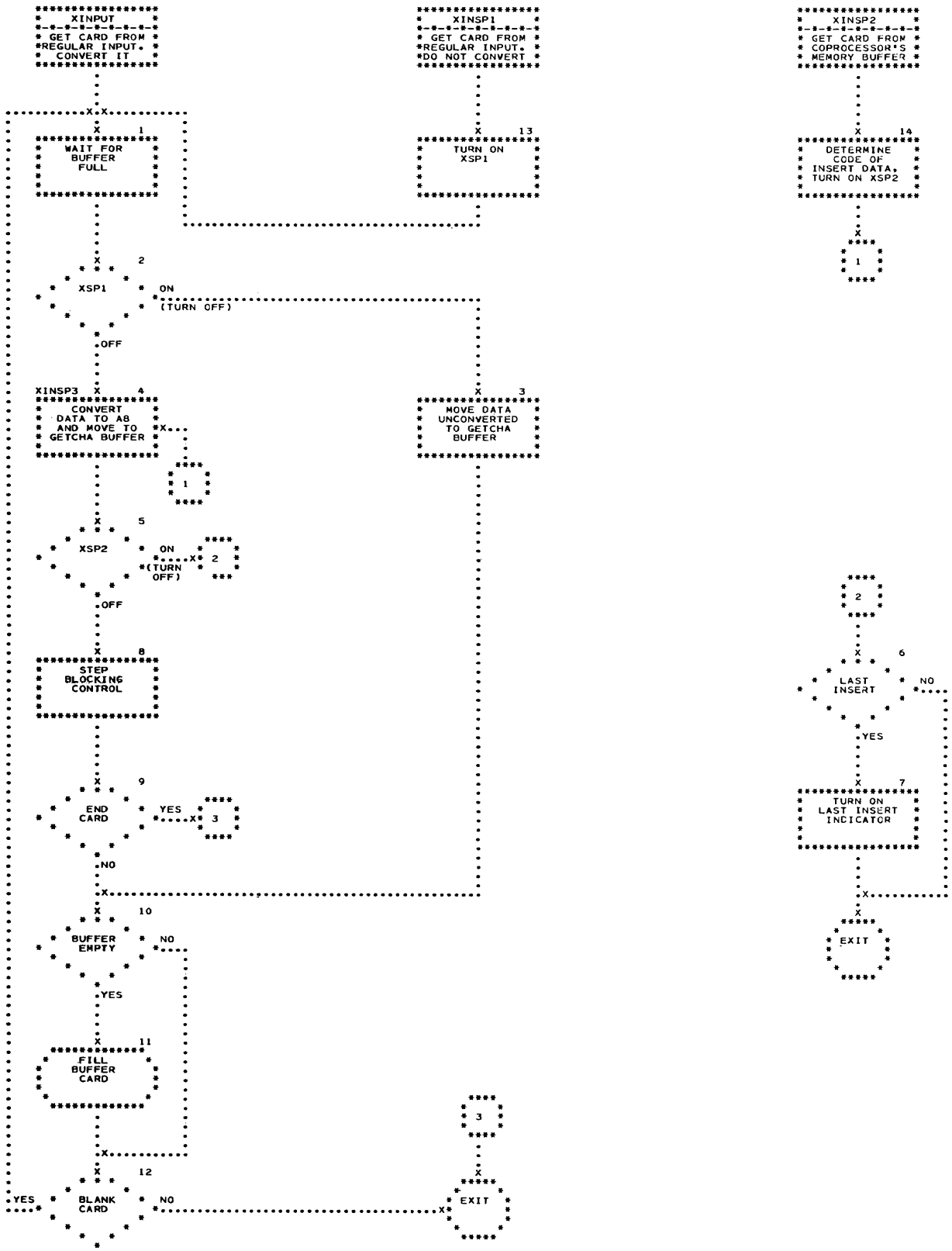
The input code specified by bits 25-27 of word 8 of COMREC is determined by digit selection. The address of the conversion routine is stored in the branch instruction following Xreada.

Boxes 5, 6

The blocking factor in bits 48-63 is stored in the count field of the blocking control (Xblkl), and in the System Input calling sequence. In the case of disk input, it is also multiplied by the number of words per card to obtain the word count of the read control word.

Box 8

The number of arcs in temporary working storage is obtained from word 1 of COMREC. One is subtracted, and this is used as the starting arc for the name file. Available arcs on the disk are divided in the ratio of 1/32; 1/32 assigned to the name file, 31/32 assigned to the intermediate file.


XINPUT
GET CARD FROM REGULAR INPUT. CONVERT IT
1
WAIT FOR BUFFER FULL
2
XSP1
ON (TURN OFF)
OFF
XINSP3
4
CONVERT DATA TO A8 AND MOVE TO GETCHA BUFFER
1
5
XSP2
ON (TURN OFF)
2
OFF
8
STEP BLOCKING CONTROL
9
END CARD
YES
3
NO
10
BUFFER EMPTY
NO
YES
11
FILL BUFFER CARD
12
BLANK CARD
YES
NO
XINSP1
GET CARD FROM REGULAR INPUT. DO NOT CONVERT
13
TURN ON XSP1
3
MOVE DATA UNCONVERTED TO GETCHA BUFFER
EXIT
XINSP2
GET CARD FROM COPROCESSOR'S MEMORY BUFFER
14
DETERMINE CODE OF INSERT DATA. TURN ON XSP2
1
2
6
LAST INSERT
NO
YES
7
TURN ON LAST INSERT INDICATOR
EXIT

LINKAGE

1. The first calling sequence to XINPUT must be: SIC, Xret; B, XINIT.

2. This usual calling sequence causes XINPUT to place at the location specified one card converted to A8 code:

```
LVI, 15, $+2
B, XINPUT
,                'Address to place card.
B,               'Return if non-IBM character
                   is encountered during CCT.
                   A8 conversion.
B,               'Return if end of file from
                   input source occurs before
                   END card is discovered.
Normal Return
```

3. This special calling sequence for the coprocessor's "want another card image" option causes XINPUT to place the next card unconverted at the address specified:

```
LVI, 15, $+2
B, Xinspl
,                'Address to place unconverted
                   card.
,                'Spacer (not used).
B,               'End-of-file return.
Normal Return
```

4. This special calling sequence to get the coprocessor's inserted card images causes XINPUT to supply the next "X" number of cards from a location other than from usual input source. ($7 must be set up as follows: VF, location of input data; CF number of cards to insert; RF, code identification number.)

```
LVI, 15, $+2
B, Xinsp2
,                'Address to place card image.
,                'Spacer (not used).
;                'Spacer (not used).
Normal Return
```

DETAILS

The input is obtained from the System Input if STRAP II is the first member of a chain, or from the disk if other than first in a chain. The input, if from the System Input, is assumed to be in IBM card code; if the input is from the disk, the address of the first arc of data, the input code, and the number of cards per arc are placed in word 8 of the Communication Record by the pre-processor. In all cases but one, the data is converted to A8 code and placed in the location specified by the calling sequence.

FLOW CHART DESCRIPTION

Box 1

The instruction---BB, Xind2, $-- is encountered to prevent processing data before the I-O operation is complete. The bit is turned off by the EOP interrupt.

Boxes 2, 3

Xspl is a bit indicating that entry was made through the ---LVI,15,$+2;B, Xinspl --- linkage. If on, a card image is given without being converted to A8 code. The blocking control value field is incremented.

Box 4

If Xspl is off, a card from the input buffer is converted to A8 code, and placed at the memory position specified.

Boxes 5 - 8

Xsp2 is a bit indicating that entry was made through the--- LVI,15,$+2;B, Xinsp2 --- linkage. If on, it is turned off. The count of $7 is checked for zero. If so, a bit is set indicating that the last card from the special source has been given, and return is made. If Xsp2 is not on, the blocking control value field is incremented.

Box 9

The card supplied is checked for the END pseudo-op. If the END card is found, return is made. If the card is blank, an indicator is set.

Box 10

The count in the blocking control is checked for zero. If zero, all cards in the input buffer have been used and the buffer must be refilled. If not zero, the blocking control count field is decremented.

Box 11

If STRAP II is first in the chain, the buffer is filled by the System Input. If not first in the chain arc address of the next block of input is obtained from the first word of the exhausted input, and the buffer is filled from the disk. Blocking control is reset.

Box 12

If the blank card indicator which may have been set in box 9 is on, the card is ignored and a branch is made to box 1. If the indicator is not on, the routine exits.

Box 13

Entrance when the next card is desired unconverted. Xsp1 is turned on.

Box 14

Entrance when a card is desired from a special input buffer. $7 is inspected to determine the memory location of the source and the number of cards in the special input. The code is determined and linkage to the proper conversion code is set up. Xsp2 is turned on.

MAJOR LOGIC AREA IN THE BETWEEN-PASSES PHASES OF STRAP II

| | | |
|---|---|---|
| UITER | ITERate over the symbol table | Iterates over the symbol table to establish a data description and value for each symbol in the table. |

---

UITER

LINKAGE

UITER inspects the symbol table twice, once between the end of Pass 1 and the beginning of Pass 2a, and again between the end of Pass 2a and the beginning of Pass 2b. Each time it is entered with a--B, UITER.

DETAILS

UITER iterates over the symbol table to establish a data description and value for each symbol. UITER also computes the memory bounds of the program being assembled.

UITER maintains a pushdown type table at the symbolic location, Utab.
The 64 bit entry in the Utab table contains: in positions 0-23 the location in the symbol table of the problem symbol and in positions 32-56 the level in the symbolic chain. An entry in the Utab table is made each time VALUE returns through Vbreak while UITER is processing a SYN chain. In coordination with the Utab table, UITER uses three index registers which have the following significance:

| Symbolic | | Field | Contents |
|---|---|---|---|
| U3 | ($3) | Value | The location in the symbol table of the base symbol in the chain |
| U4 | ($4) | Value | The location of the latest entry made in the Utab table. |
| | | Count | The current level of evaluation. |
| U9 | ($5) | Value | The location in the symbol table of the symbol currently being evaluated by UITER. |

FLOW CHART DESCRIPTION *

After initialization we determine whether or not we are in Pass 1. (Np2ind is zero if we are in Pass 1.)

Boxes 7, 15 (after Pass 1)

Get each symbol table entry by branching to ANEXT. When there are no more symbol table entries, leave UITER.

Box 8

Space to location of function in order to investigate the entry. The delete mark, the field indicating the length of the entry, the ID field, the field indicating the length of the remainder of the argument, and the remainder of the argument precede the function field in the vle.

Box 9

Branch-on-bit-and-set-to-one Zst0 (everything defined and computed) and then go to ANEXT to get the next symbol table entry. By "everything" it is meant value, dds, and dimension properties.

Boxes 16, 31

Branch-on-zero-bit Zst3 (existence of dds reference has been decided). By dds reference address, it is meant a location to find the address of a dds.

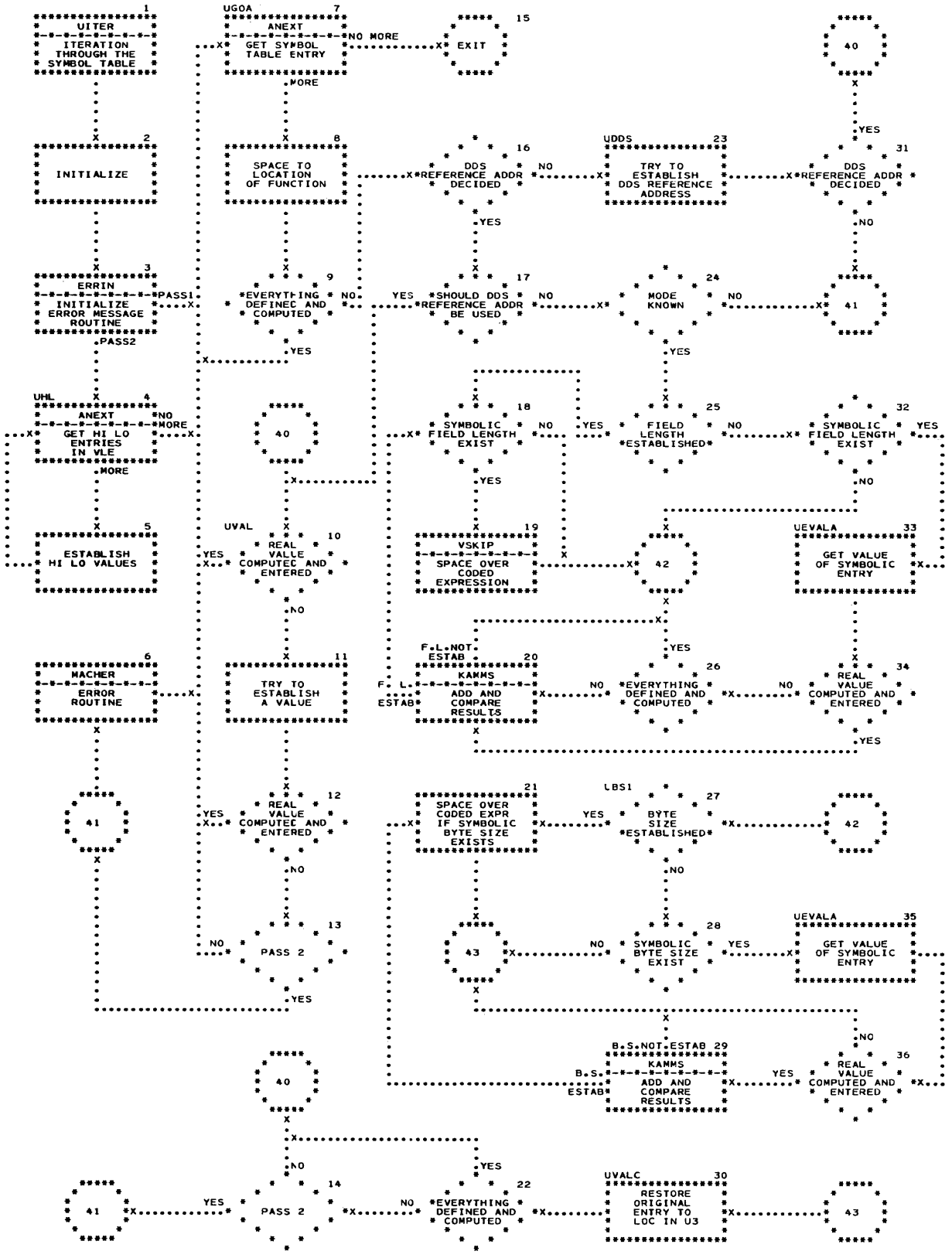
Box 23

Test for circular value by seeing if Zst17 (processing bit for mode) is on. If value is circular, give reference address of zero. Branch to VALUE subroutine to establish dds reference address.

Box 17

Branch-on-bit Zst32 (use dds reference address) to box 10. The dds reference address is only used if the dds is implicit.

*Flow chart of UITER on next page


UITER
ITERATION THROUGH THE SYMBOL TABLE
INITIALIZE
ERRIN
INITIALIZE ERROR MESSAGE ROUTINE
PASS1
PASS2
ANEXT
GET HI LO ENTRIES IN VLE
MORE
NO
ESTABLISH HI LO VALUES
MACHER
ERROR ROUTINE
UGOA
ANEXT
GET SYMBOL TABLE ENTRY
NO MORE
SPACE TO LOCATION OF FUNCTION
EVERYTHING DEFINED AND COMPUTED
UVAL
REAL VALUE COMPUTED AND ENTERED
TRY TO ESTABLISH A VALUE
PASS 2
EXIT
DDS REFERENCE ADDR DECIDED
SHOULD DDS REFERENCE ADDR BE USED
SYMBOLIC FIELD LENGTH EXIST
VSKIP SPACE OVER CODED EXPRESSION
F.L.NOT.ESTAB
KAMMS ADD AND COMPARE RESULTS
SPACE OVER CODED EXPR IF SYMBOLIC BYTE SIZE EXISTS
UDDS
TRY TO ESTABLISH DDS REFERENCE ADDRESS
MODE KNOWN
FIELD LENGTH ESTABLISHED
LBS1
BYTE SIZE ESTABLISHED
SYMBOLIC BYTE SIZE EXIST
B.S.NOT.ESTAB
UVALC
RESTORE ORIGINAL ENTRY TO LOC IN U3
UEVALA
GET VALUE OF SYMBOLIC ENTRY
40
41
42
43
YES
NO
MORE
PASS1
PASS2
ESTAB
B.S.

Box 24

Branch-on-bit Zst10 (mode known) to box 25.

Boxes 10, 12

Branch-on-bit Zst26 (real value computed and entered). By real value it is meant that we know the value of a symbol.

Box 11

Test for circular value. If circular value, assign a value of zero; otherwise branch to VALUE to attempt to establish a value. In the case of a DR whose dimension product has not yet been formed, branch to VDIMEN. If the product of dimensions is not formed by VDIMEN, turn off Zst0, and continue.

Box 13

Branch-on-zero-bit Np2ind (Pass 2 indicator) to box 7. The value should be established by the end of Pass 2a; if it is not, branch to MACHER (error subroutine).

Box 25

Branch-on-bit Zst11 (field length has been established) to box 18.

Box 32

Branch-on-bit Zst12 (symbolic field length exists) to box 33.

Box 18

Branch-on-zero-bit Zst12 to box 27.

Box 19

Branch to VSKIP subroutine to skip the pointer over the coded expression for the field length because the field length has already been established.

Box 20

Branch to Kamms procedure in DECODE to test whether the field length was properly specified, i.e., to see if it is not more than 64 bits and not in bit style. An illegal condition causes both an error message to be given and the field length to be truncated. (Bit and integer values are combined.)

Box 27

Branch-on-bit-and-set-to-one Zst0 (everything defined and computed) to box 27.

Boxes 33 and 34

Described under boxes 11 and 10, respectively.

Box 27

Branch-on-bit Zst14 (byte size has been established) to box 27.

Box 28

Branch-on-bit Zst15 (symbolic byte size exists) to box 35.

Box 21

Branch to box 30 if symbolic byte size does not exist. If symbolic byte size does exist, branch to VSKIP subroutine to skip the pointer over the coded expression for the byte size because of the byte size has already been established.

Boxes 35 and 36

Described under boxes 11 and 10, respectively.

Box 29

Branch to Kamms procedure in DECODE to test whether the byte size was properly specified, i.e., to see if it is not more than 8 bits and not in bit style. An illegal condition causes both an error message and the byte size to be truncated. (Bit and integer values are combined.)

Box 22

Branch-on-bit-and-set-to-one Zst0 (described under box 9) to box 10.

Box 14

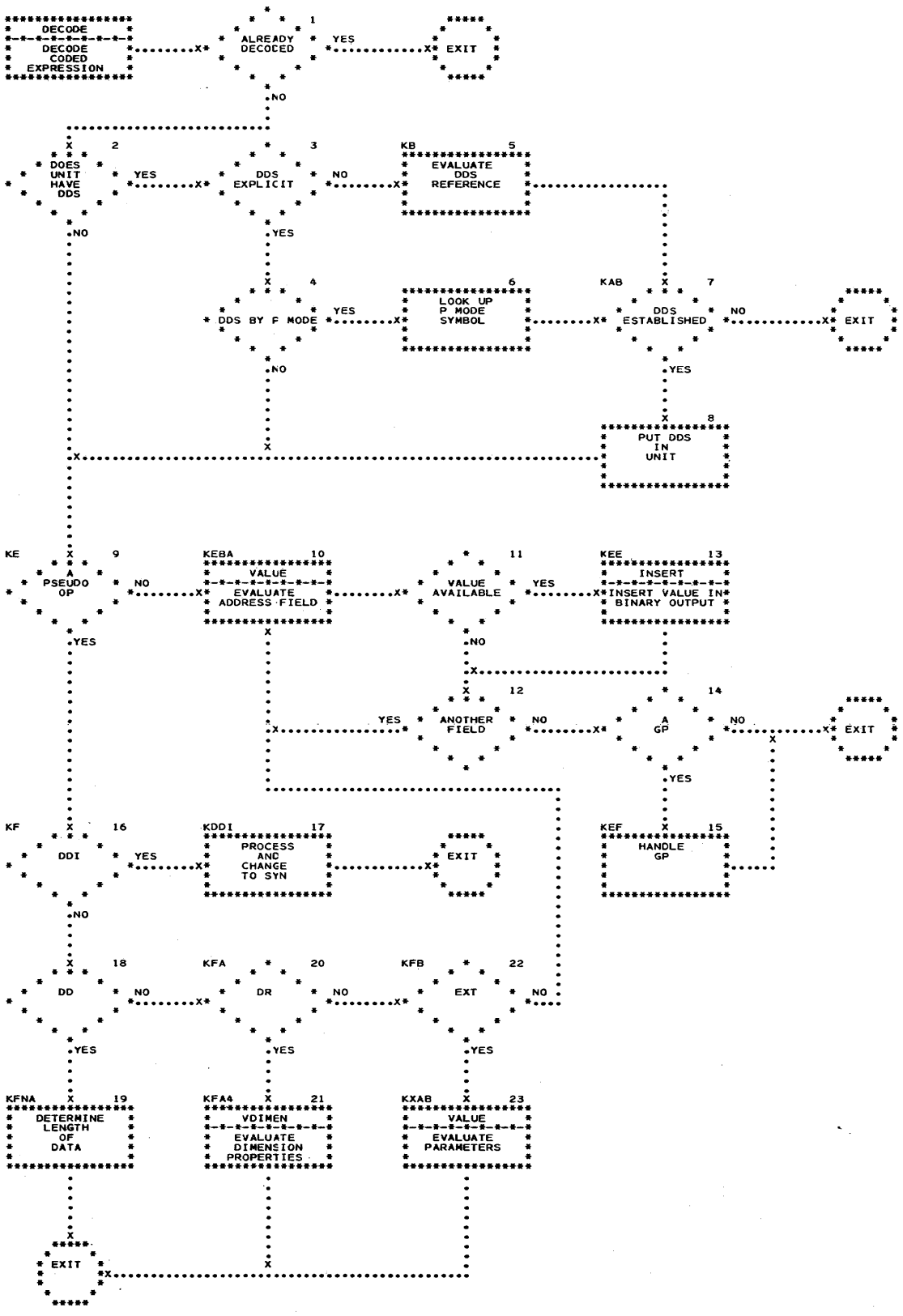
Branch-on-zero-bit Np2ind (Pass 2 indicator) to box 10

Boxes 4, 5, (after Pass 2a)

Branch to ANEXT subroutine to get each hi-lo entry made by Pass 2 in the vle table; establish the memory bounds of the assembly.

MAJOR LOGIC AREAS IN THE PASS 2 PHASES OF STRAP II

| NPS2 | Main Flow for Pass 2 | Includes the Processing in the Following Subroutines and Subprocedures |
|---|---|---|
| DECODE | DECODE coded expressions | Resolves the coded expressions in the expanded instruction units. |
| INTIN | INput the INTermediate expanded instruction unit | Reads the intermediate expanded instruction units from disk for processing during Pass 2 and writes them back on disk. |
| INSERT | INSERT values into the binary output skeleton | Performs the necessary address arithmetic on the bit and integer values provided by VALUE, and inserts the results into the correct field of the binary output skeleton. |
| NEXT | Pass 2's EXTract procedure | Has the expressions of the parameters evaluated and saves the values; adjusts the companion unit accordingly with the proper updating of the location counter. |
| NMCP | Pass 2's MCP procedure | Completes the symbolic card image in each MCP expanded instruction unit for the later punching and printing by OUTPUT. |
| OUTPUT | OUTPUT the final documents | - - - - - - |
| NAMEIN | Bring a NAME IN off the name file | Fetches each name to be printed with its instruction on the listing. |
| ERRNUM | Get the ERRor page and line NUMber | Computes the page and line number for each entry in the five lists of detected programmer errors; puts the second and fourth lists in order according to page and line number. |
| OPPUN | Output's PUNching procedure | Puts the STRAP II binary on an output device. |
| OEDIT | Output's EDITing procedure | Sets up the carriage control character for each line to be printed; provides both the heading which appears on the top of each page and the total line message. |
| OPLIST | Output's LISTing procedure | Puts the listing on an output device. |
| NQBTP | PUNSYM procedure | Creates a card image for each symbol specified individually by a PUNSYM or collectively by a PUNALL. |
| NUNDSY | Pass 2's UNDefined SYmbol listing | Sets up and has printed the lists of symbols that are undefined, multiply-defined, circularly defined, and the ones never used in the program. |
| ERRPRT | PRinT ERRor messages | - - - - - - |


DECODE
DECODE CODED EXPRESSION
1
ALREADY DECODED
YES
EXIT
NO
2
DOES UNIT HAVE DDS
YES
NO
3
DDS EXPLICIT
NO
YES
KB 5
EVALUATE DDS REFERENCE
4
DDS BY P MODE
YES
NO
6
LOOK UP P MODE SYMBOL
KAB 7
DDS ESTABLISHED
NO
EXIT
YES
8
PUT DDS IN UNIT
KE 9
A PSEUDO OP
NO
YES
KE8A 10
VALUE EVALUATE ADDRESS FIELD
11
VALUE AVAILABLE
YES
NO
KEE 13
INSERT INSERT VALUE IN BINARY OUTPUT
12
ANOTHER FIELD
YES
NO
14
A GP
NO
EXIT
YES
KF 16
DDI
YES
NO
KDDI 17
PROCESS AND CHANGE TO SYN
EXIT
KEF 15
HANDLE GP
18
DD
NO
YES
KFA 20
DR
NO
YES
KFB 22
EXT
NO
YES
KFNA 19
DETERMINE LENGTH OF DATA
KFA4 21
VDIMEN EVALUATE DIMENSION PROPERTIES
KXAB 23
VALUE EVALUATE PARAMETERS
EXIT

LINKAGE

DECODE is entered from pass 2a and 2b if Zifp2 is one.

On Entry

The calling sequence is: SIC, Decodz; B, DECODE. Zifall is set to one on entry by DECODE.

On Exit

If DECODE is unable to completely assemble the binary output, Zifall is set to zero before returning.

FLOW CHART DESCRIPTION

Box 1

Establish if DECODE is finished with unit by testing if Zifall is on.

Box 2

Test Zoddsa of the operation question bits to determine if a dds is needed for this unit.

Boxes 3, 4

Test Zifdds and Zifpm of the expanded instruction unit to determine if the dds is explicit or implicit and if it is specified by a P-mode.

Box 5

Handle case when dds needed but none given explicitly, made up according to convention:

1. If op is ambiguous (variable field length or floating point), it is compiled as (BU, 64, 8).
2. If op is clearly variable field length it is compiled as (BU, 64, 8).
3. If op is clearly variable field length and an immediate, it is compild as (BU, 24, 8).
4. If op is only floating point except for E or E+I, then it is compiled as normalized; if op is E or E+I, then it is compiled as unormalized.

Box 6

Using the Zst24 subroutine SEARCH look up the P-mode symbol. Turn the Zst24 of the symbol table entry to mark as used.

Boxes 7, 8

If the dds has been established, insert it in the dds field of the expanded instruction unit.
If the dds has not been established, turn Zifall off and return.

Box 9

Test Zopo to determine if it is a pseudo-op.

Boxes 14, 15

Test for general parenthetical expressions, if there is a GP evaluate and insert it in the binary output.
If unable to evaluate, turn Zifall off and return.

Box 13

INSERT is used to fill the major fields in the binary output word of the expanded instruction unit.

Boxes 10-12

Use VALUE to evaluate all major fields.

Boxes 16, 17

If a DDI, process and change to SYN and return. DECODE uses part of OUTPUT to evaluate the DDI, and OR's it into the expanded instruction unit.

Boxes 18, 19

If an DD, multiply field length by number of fields, store result in the expanded instruction unit, and return.

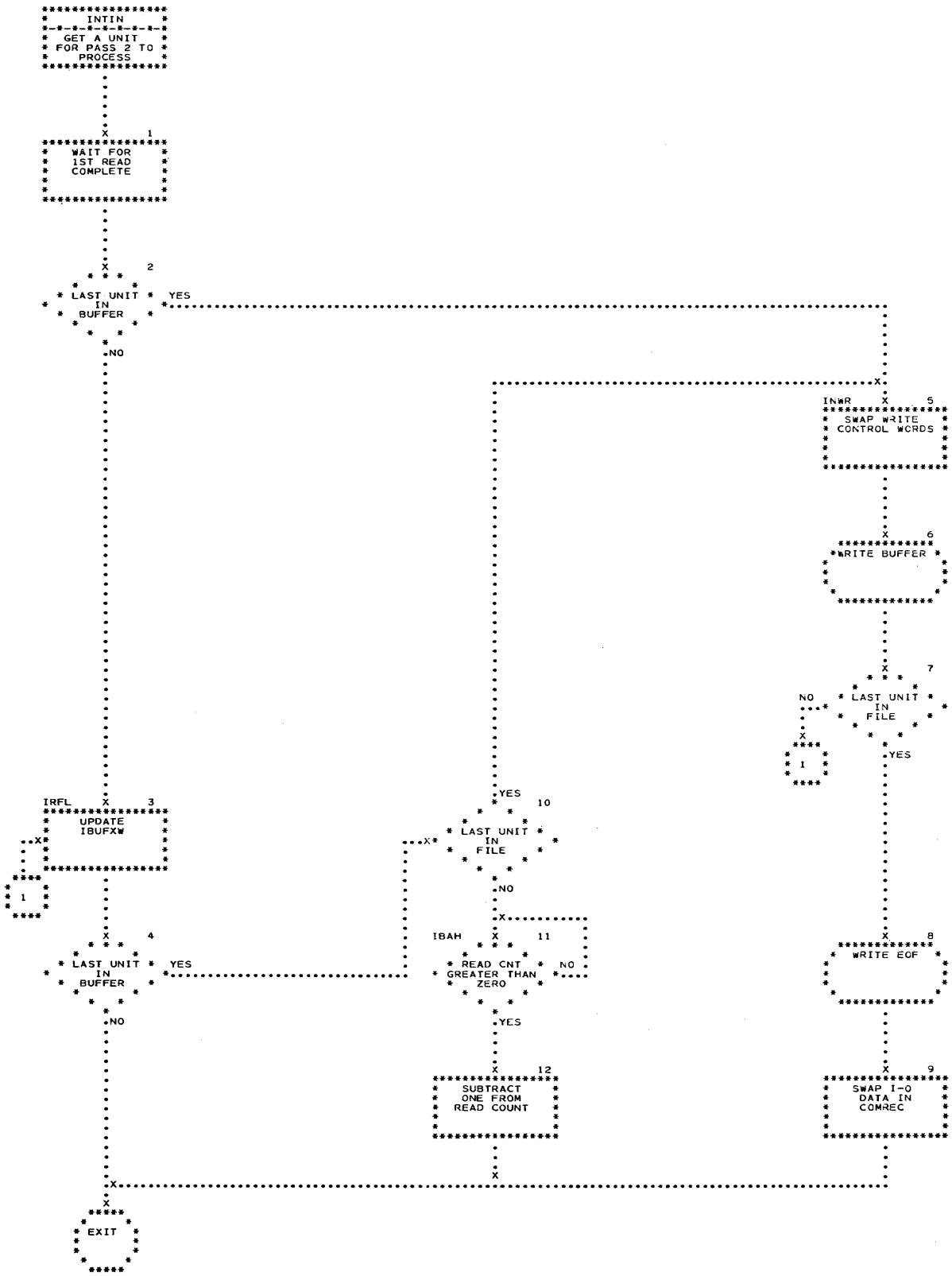
Boxes 20, 21

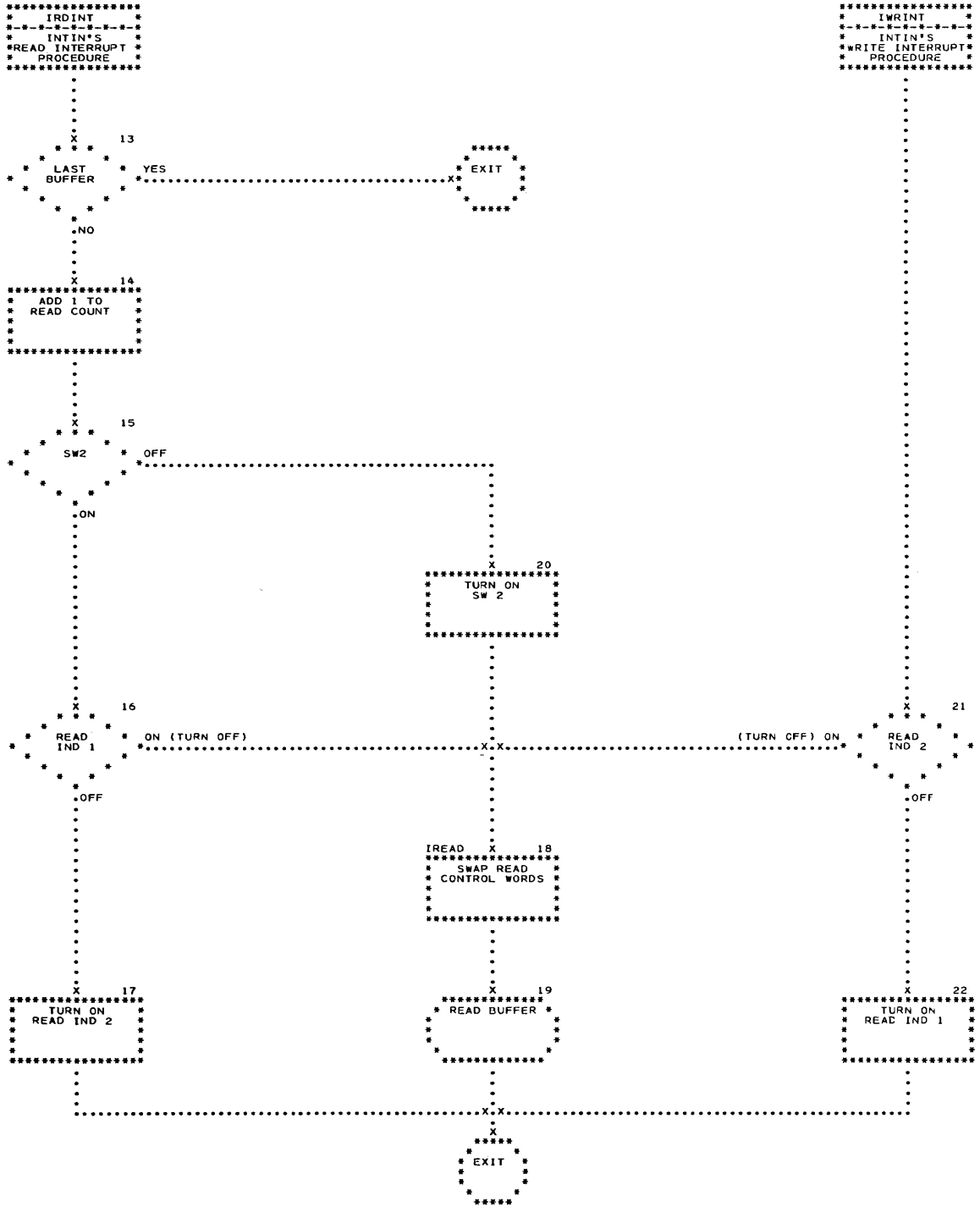
If an DR, evaluate the dimension properties; multiply the dimension by the field length to reserve the amount of space required. The result is stored in the expanded instruction unit.

Boxes 22, 23

GETFLD goes to VALUE to get values, and if available, stores them for use by the extract section of Pass 2.

If the op is not DDI, DD, DR, or EXT, GETFLD returns for normal insertion of address field.


INTIN
GET A UNIT FOR PASS 2 TO PROCESS
1
WAIT FOR 1ST READ COMPLETE
2
LAST UNIT IN BUFFER
YES
NO
INWR
5
SWAP WRITE CONTROL WORDS
6
WRITE BUFFER
7
LAST UNIT IN FILE
NO
YES
1
IRFL
3
UPDATE IBUFXW
1
4
LAST UNIT IN BUFFER
YES
NO
YES
10
LAST UNIT IN FILE
NO
IBAH
11
READ CNT GREATER THAN ZERO
NO
YES
12
SUBTRACT ONE FROM READ COUNT
8
WRITE EOF
9
SWAP I-O DATA IN COMREC
EXIT


IRDINT
INTIN'S
READ INTERRUPT
PROCEDURE
IWRINT
INTIN'S
WRITE INTERRUPT
PROCEDURE
13
LAST BUFFER
YES
EXIT
NO
14
ADD 1 TO READ COUNT
15
SW2
OFF
ON
20
TURN ON SW 2
16
READ IND 1
ON (TURN OFF)
(TURN OFF) ON
21
READ IND 2
OFF
OFF
IREAD
18
SWAP READ CONTROL WORDS
17
TURN ON READ IND 2
19
READ BUFFER
22
TURN ON READ IND 1
EXIT

LINKAGE

On Entry

1. Initially the calling sequence is:

SIC, Intiz; B, Ininit.

2. Subsequently the calling sequence is:

SIC, Intiz; B, INTIN.

On Exit

The control word preceding the expanded unit is placed in Ibufxw for the user routine.

FLOW CHART DESCRIPTION

Box 1

A -- BEW, $ -- is encountered which prevents processing the first buffer before the read operation has been completed. The EOP interrupt fix-up will change the BEW to a NOP.

Box 2

Bit 25 of Ibufxw is tested for a one. If one, the control word is the last in a buffer, and the buffer is ready to be written on the disk.

Box 3

The next control word is placed in Ibufxw by refilling $2 and storing.

Box 4

The control word is tested to determine if it is the last in the buffer. If not, return is made.

Boxes 5, 6

The write control words (there are three, one for each buffer) are rotated, and the buffer is written on the disk.

Boxes 7-9

Bit 26 of the control word is checked for a one indicating the last control word in the file. If so, an additional buffer is written indicating end of file.

Boxes 10-12

If Bit 26 is off in the control word, the read counter is checked for a value greater than zero. If not greater, the program must wait until a read operation is completed. The EOP fix-up will add one to the read counter. When the counter is greater than zero, one is subtracted from it, and return is made.

Box 13

The buffer just read is checked for the end-of-file record. If so, return is made thru $RET.

Boxes 14, 15, 20

One is added to the read counter. If switch 2 is off (first time only), it is turned on.

Boxes 18, 19

Read control words are swapped, and another buffer is read. Return is made thru $RET.

Boxes 16, 17

Initially on. If on, it is turned off, and a read is initiated. If off, read indicator 2 is turned on, thus permitting the next write interrupt to initiate the read.

Boxes 21, 22

Entered as a result of a write interrupt. If on, read indicator 2 is turned off, and initiates a read. If off, read indicator 1 is turned on.

LINKAGE

DECODE uses any of the INSERT subroutines by the calling sequence: SIC, Sgez; B, an address of one of the INSERT subroutines. The address in the Branch instruction is determined by the current fill index word.

DETAILS

After DECODE has a coded expression, which is in the unit, evaluated by VALUE, a value must be inserted into the appropriate field in the binary output.

For each type of instruction, there has been set up a chained set of index words so DECODE can successively use the appropriate INSERT subroutine after getting each coded expression in the unit evaluated by VALUE. (Recall that MAIN puts the location of the set of fill index words for this type of op in the unit after the look-up in the primary operation table.)

The format of each of the fill index words is: Value field contains the address of the INSERT subroutine to put value in particular field of the binary output skeleton in the unit. Count field contains the number of coded expressions each of whose values will be given to this particular INSERT subroutine. (This count field is one in almost all type of instructions, INDMK and LVS being the exceptions.) Refill field contains the location of the index word to be used in filling in the value of the next field in the binary output skeleton.

Thus, for example, the field types list for a variable field length operation would be -- XW, S24A, 1, $ + 1; XW, SOFF, 1, 0 -- while for a LVS it would be -- XW, SJFLD, 1, $ + 1; XW, SBIT, 16, 0 .

Algebraic addition of bit and integer addresses is performed according to the type of field into which the sum is to be inserted. For example, if the address SAM + 10 is to be inserted into an 18-bit address, the routine adds 10 full words to the address of SAM; if a 19-bit address, it adds 10 half words; if a 24-bit address, it adds 10 bits. Note that the addition is determined by the function of the address, not the size of the field. To fill the addresses in the instruction TI, 7, SAM + 10, PAT + 10, the routine adds 10 full words to both addresses. Although SAM + 10 is a 24-bit field and PAT + 10 is a 19-bit field, both function as 18-bit addresses.

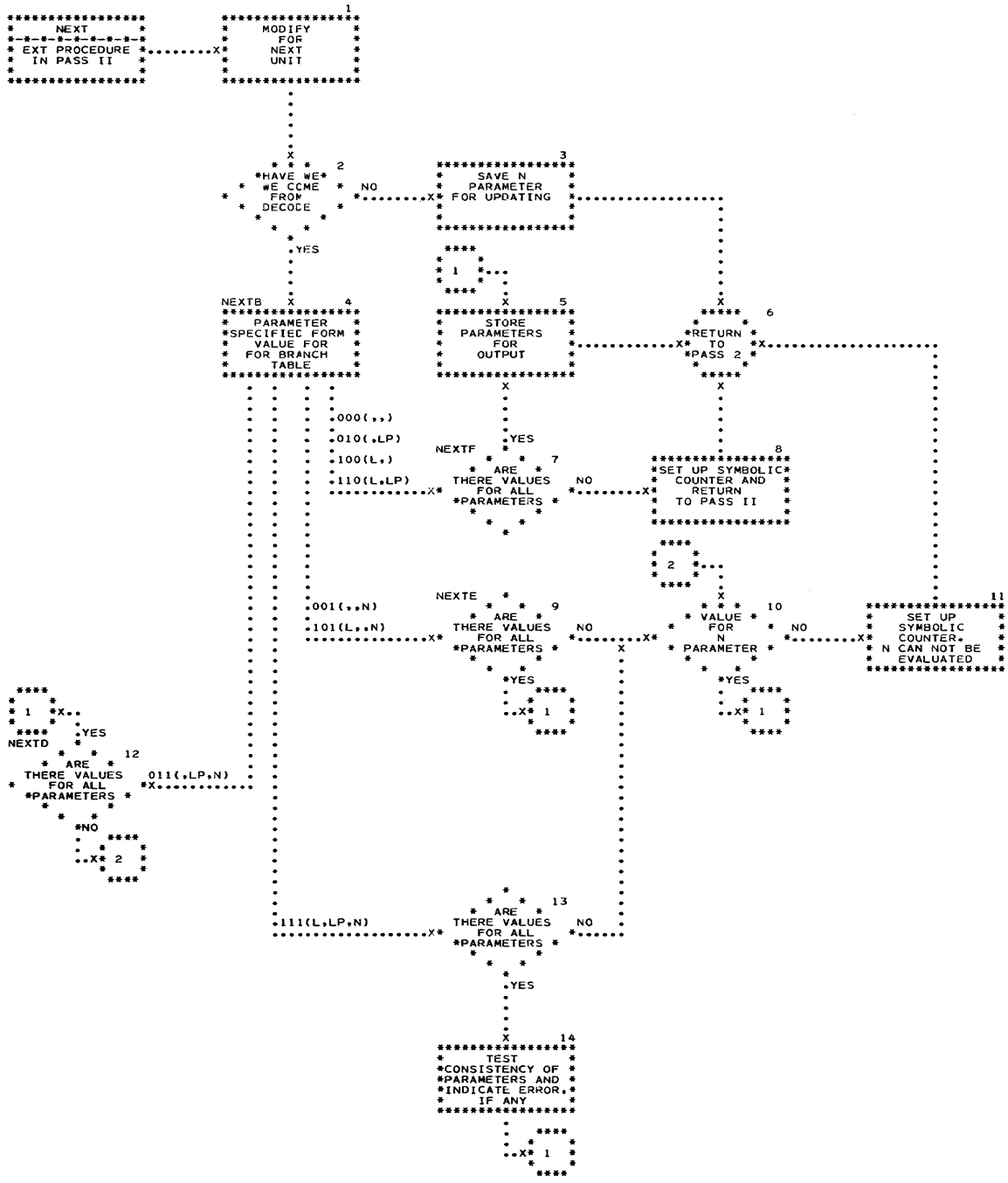
A two's complement is formed to take the place of a negative result intended for an unsigned field. Bits are truncated where necessary to fit the field, and the appropriate error messages are executed. INSERT gets the absolute value of the coded fields from three locations:

Sinb: bit value.
Sini: integer value.
Sinx: index value.

NOTE: In the case of immediate addresses, it should be noted that the immediate address is filled into the word according to the given field length, unless it contains a bit address or the given field length exceeds 24, in which cases the field is filled in as a 24-bit address.

The INSERT subroutines are:

| | |
|---|---|
| S18A | 18-bit address and I field. |
| S18NI | 18-bit address, no index allowed. |
| S19NI | 19-bit address, no index allowed. |
| S19KF | 19-bit address, K field |
| S19IA | 19-bit address, and I field. |
| S1918A | 19-bit address and I field, error-flagged if 19th bit not zero. |
| S24A | 24-bit address and I field. |
| STTI | 24-bit address and I field, error-flagged if more than 18 bits are specified. |
| SSIC24 | 24-bit address and I field, error-flagged if more than 19 bits are specified. |
| S24SNI | 24-bit signed field with index if any. |
| S24CW | 24-bit signed field, error-flagged if more than 18 bits are specified. |
| SFLGS | Flag bits 25-27. |
| S19BBK | 19-bit branch address and K field. |
| STT19A | 19-bit transmit address and I field, error-flagged if more than 18 bits are specified in address. |
| S18CF | 18-bit count field. |
| S18RF | 18-bit refill field. |
| SOFF | Offset and I field. |
| SCHAN | 7-bit channel address and I field. |
| SXP10 | Signed exponent field or normal fraction shift. |
| SBIT | Index indicator bits for LVS instruction. |
| SJFLD | J field. |
| STJFLD | J field for transmit and swap instructions. |
| SIMAD | Immediate addresses. |
| SINDMK | Indicator mask. |


NEXT
EXT PROCEDURE
IN PASS II
1
MODIFY FOR NEXT UNIT
HAVE WE COME FROM DECODE
NO
YES
2
SAVE N PARAMETER FOR UPDATING
3
NEXTB
PARAMETER SPECIFIED FORM VALUE FOR FOR BRANCH TABLE
4
STORE PARAMETERS FOR OUTPUT
5
RETURN TO PASS 2
6
.000(,,)
.010(,LP)
.100(L,)
.110(L,LP)
NEXTF
ARE THERE VALUES FOR ALL PARAMETERS
7
SET UP SYMBOLIC COUNTER AND RETURN TO PASS II
8
.001(,,N)
.101(L,,N)
NEXTE
9
VALUE FOR N PARAMETER
10
SET UP SYMBOLIC COUNTER. N CAN NOT BE EVALUATED
11
NEXTD
12
011(,LP,N)
.111(L,LP,N)
13
TEST CONSISTENCY OF PARAMETERS AND INDICATE ERROR, IF ANY
14

NEXT

LINKAGE

Pass 2 enters the extract procedure from the digit select pseudo-op table.

FLOW CHART DESCRIPTION

Box 1

Modify Pass 2 to skip rounding the location counter, and branch to a special location counter updating. Also modify Pass 2 to stop the extract of a pseudo-op.

Box 2

Check to see if DECODE has already handled the unit. Zifp2 is always on in Pass 2.

Box 3

Store N value in Zilbo for updating, and return.

Box 4

Set up branch to determine what fields are given.

Boxes 7, 9, 12, 13

Determine if Zifall is on to verify that there are values for all parameters.

Box 5

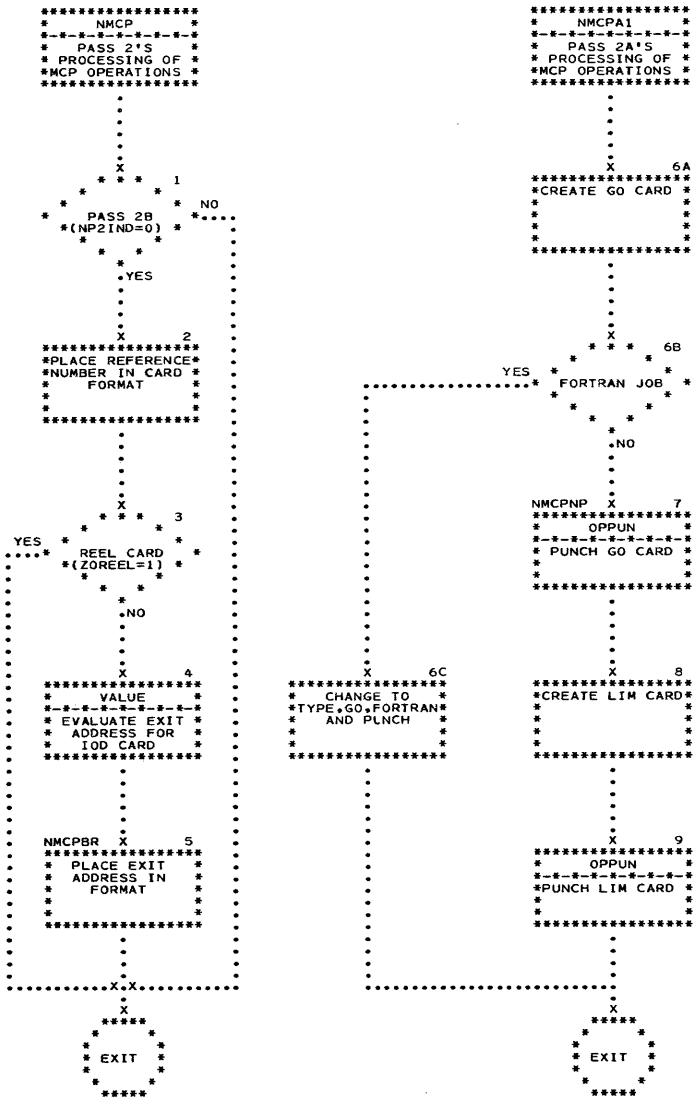
The value of N has been determined; insert this value in the expanded instruction unit, and return.

Box 8

Set up symbolic counter when the N parameter is not specified and when the other parameters cannot be evaluated at this point.

Boxes 10, 11

Set up symbolic counter. The expression for N cannot be evaluated at this point.


NMCP
PASS 2'S
PROCESSING OF
MCP OPERATIONS
1
PASS 2B
(NP2IND=0)
NO
YES
2
PLACE REFERENCE
NUMBER IN CARD
FORMAT
3
REEL CARD
(ZOREEL=1)
YES
NO
4
VALUE
EVALUATE EXIT
ADDRESS FOR
IOD CARD
NMCPBR
5
PLACE EXIT
ADDRESS IN
FORMAT
EXIT
NMCPA1
PASS 2A'S
PROCESSING OF
MCP OPERATIONS
6A
CREATE GO CARD
6B
FORTRAN JOB
YES
NO
NMCPNP
7
OPPUN
PUNCH GO CARD
8
CREATE LIM CARD
9
OPPUN
PUNCH LIM CARD
EXIT
6C
CHANGE TO
TYPE,GO,FORTRAN
AND PUNCH

## LINKAGE

At the beginning of Pass 2b for each Master Control program operation, the NMCP subroutine is entered to complete the card images for later punching.

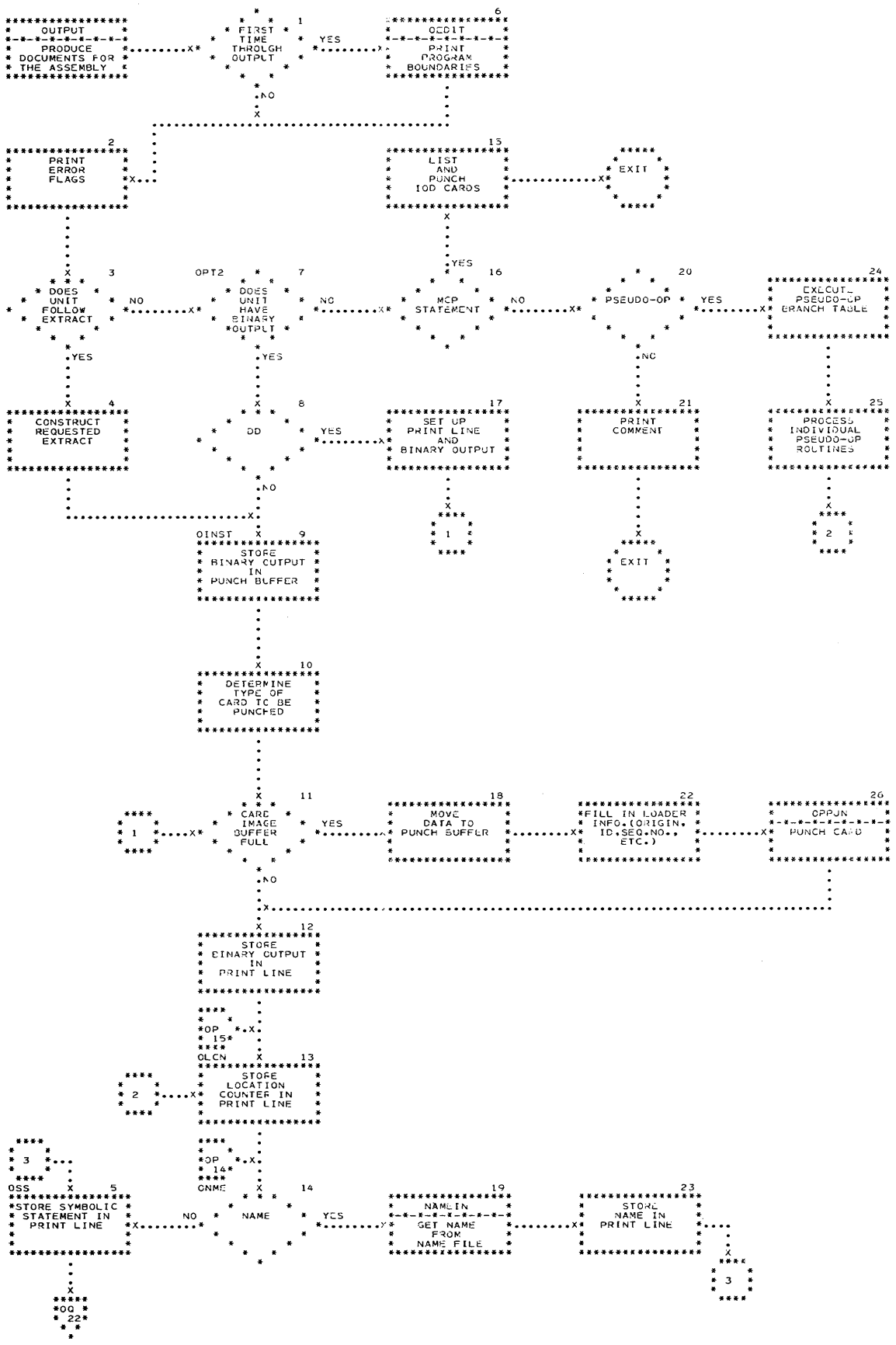
## FLOW CHART DESCRIPTION

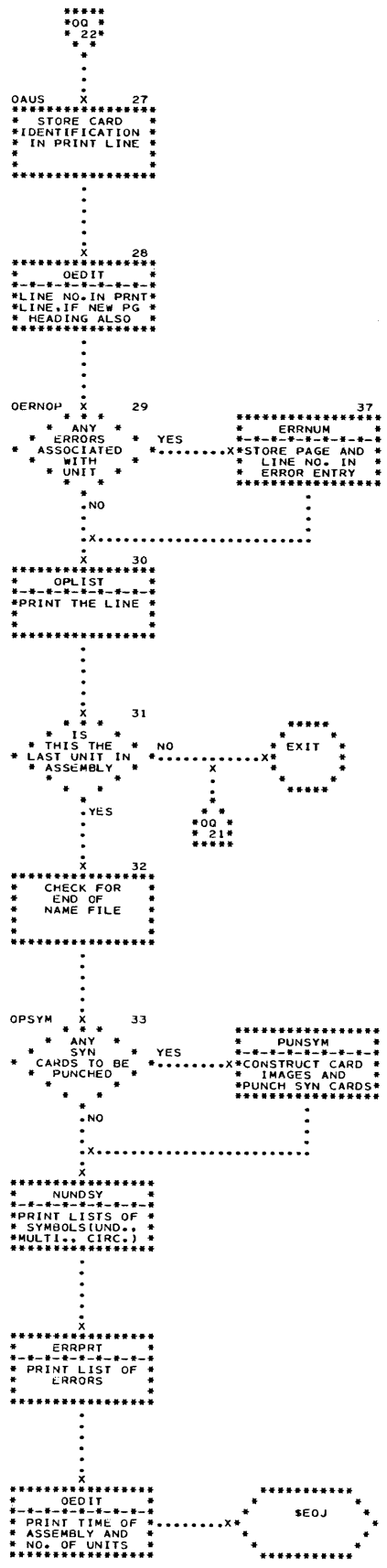
### Box 1

During Pass 2a the master control program units do not receive any attention.

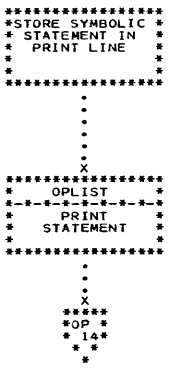
### Boxes 3-5

Primarily NMCP must complete the eighty column A8 card image in the unit. The two fields that must be supplied in IOD card image are the reference number field and the I-O table of exits field. The source of the reference number is a binary counter in the fixed portion of the unit; the source of the I-O table of exits address is a coded expression in the variable portion of the unit. Once evaluated, both values are converted to A8 code and inserted into the card image saved in the unit. If no table of exit address had been specified in the source input, NMCP leaves the field blank.

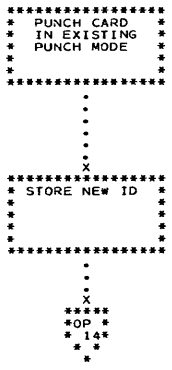

OUTPUT
PRODUCE DOCUMENTS FOR THE ASSEMBLY
FIRST TIME THROUGH OUTPUT
YES
NO
CREDIT PRINT PROGRAM BOUNDARIES
6
PRINT ERROR FLAGS
2
LIST AND PUNCH IOD CARDS
13
EXIT
DOES UNIT FOLLOW EXTRACT
3
NO
YES
OPT2
DOES UNIT HAVE BINARY OUTPUT
7
MCP STATEMENT
16
PSEUDO-OP
20
EXECUTE PSEUDO-OP BRANCH TABLE
24
CONSTRUCT REQUESTED EXTRACT
4
DD
8
SET UP PRINT LINE AND BINARY OUTPUT
17
PRINT COMMENT
21
PROCESS INDIVIDUAL PSEUDO-OP ROUTINES
25
1
2
OINST
STORE BINARY OUTPUT IN PUNCH BUFFER
9
DETERMINE TYPE OF CARD TO BE PUNCHED
10
CARD IMAGE BUFFER FULL
11
MOVE DATA TO PUNCH BUFFER
18
FILL IN LOADER INFO.(ORIGIN, ID,SEQ.NO., ETC.)
22
OPPUN PUNCH CARD
26
STORE BINARY OUTPUT IN PRINT LINE
12
*OP *.X.
* 15*
OLCN
STORE LOCATION COUNTER IN PRINT LINE
13
*OP *.X.
* 14*
ONME
OSS
STORE SYMBOLIC STATEMENT IN PRINT LINE
5
NAME
14
NAMEIN GET NAME FROM NAME FILE
19
STORE NAME IN PRINT LINE
23
3
*OQ *
* 22*


*0Q *
* 22*
OAUS X 27
STORE CARD IDENTIFICATION IN PRINT LINE
X 28
OEDIT
LINE NO. IN PRNT LINE, IF NEW PG HEADING ALSO
OERNOP X 29
ANY ERRORS ASSOCIATED WITH UNIT
YES
ERRNUM
STORE PAGE AND LINE NO. IN ERROR ENTRY
NO
X 30
OPLIST
PRINT THE LINE
X 31
IS THIS THE LAST UNIT IN ASSEMBLY
NO
EXIT
YES
*0Q *
* 21*
X 32
CHECK FOR END OF NAME FILE
OPSYM X 33
ANY SYN CARDS TO BE PUNCHED
YES
PUNSYM
CONSTRUCT CARD IMAGES AND PUNCH SYN CARDS
NO
X
NUNDSY
PRINT LISTS OF SYMBOLS(UND., MULTI., CIRC.)
X
ERRPRT
PRINT LIST OF ERRORS
X
OEDIT
PRINT TIME OF ASSEMBLY AND NO. OF UNITS
$EOJ

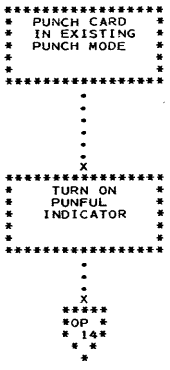
PRINID



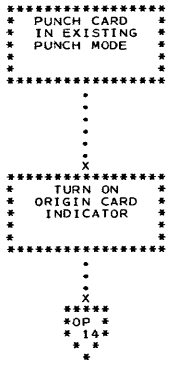

PUNID



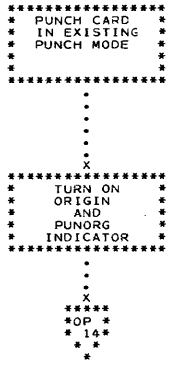

PUNFUL



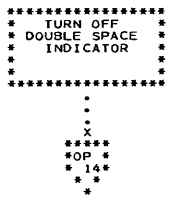

PUNNOR



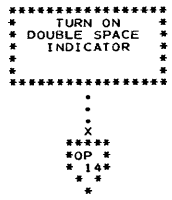

PUNORG



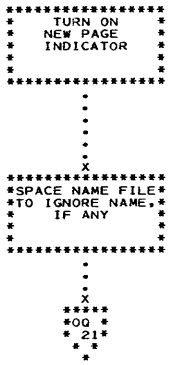

PRNS



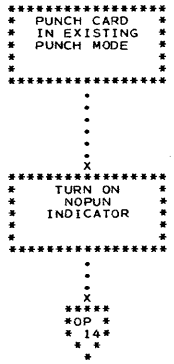

PRND



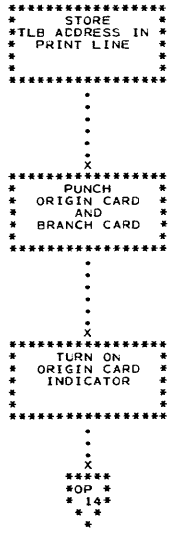

SKIP

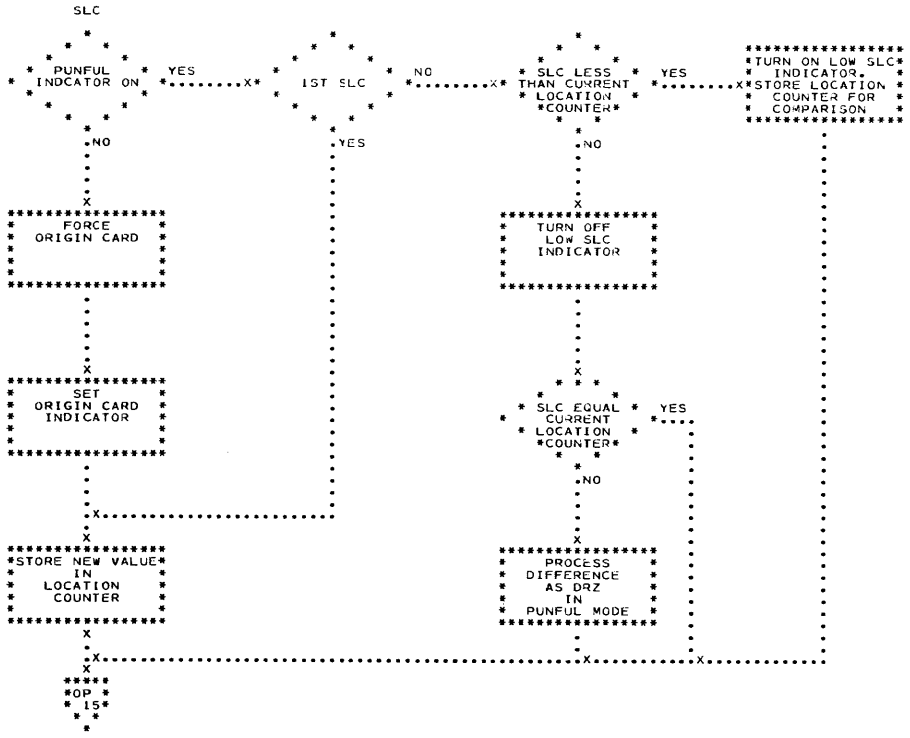



NOPUN

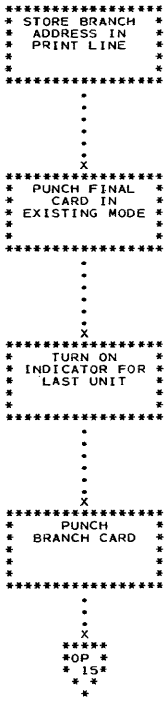



TLB

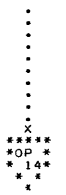



SLC
PUNFUL INDCATOR ON
YES
1ST SLC
NO
SLC LESS THAN CURRENT LOCATION *COUNTER*
YES
*TURN ON LOW SLC* INDICATOR. *STORE LOCATION COUNTER FOR COMPARISON
NO
YES
NO
FORCE ORIGIN CARD
TURN OFF LOW SLC INDICATOR
SET ORIGIN CARD INDICATOR
SLC EQUAL CURRENT LOCATION *COUNTER*
YES
NO
*STORE NEW VALUE* IN LOCATION COUNTER
PROCESS DIFFERENCE AS DRZ IN PUNFUL MODE
*OP * *IS*

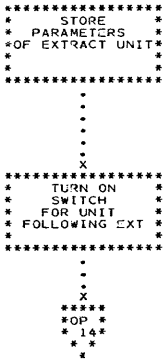
END



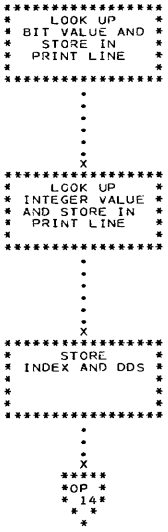

SLCR



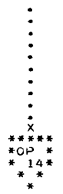

EXT



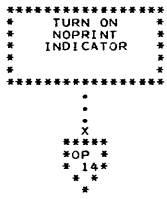

SYN



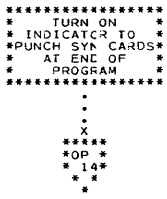

DDI



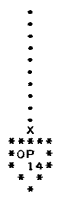

NOPRNT



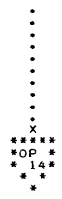

PUNSYM



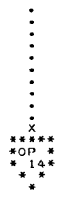

SEM



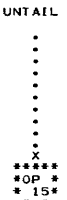

REM



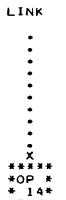

.TAIL



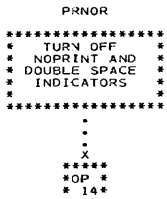

UNTAIL



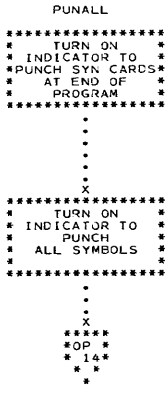

LINK



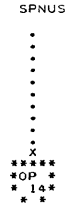

PRNOR

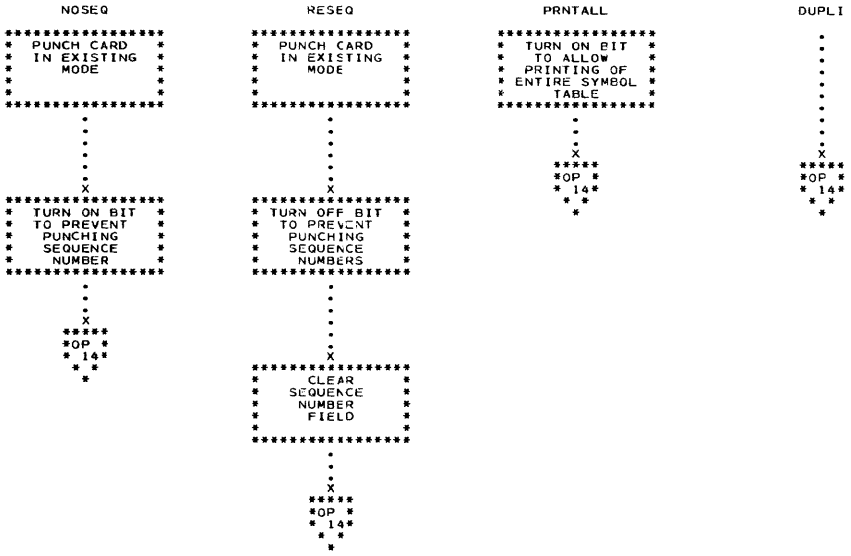


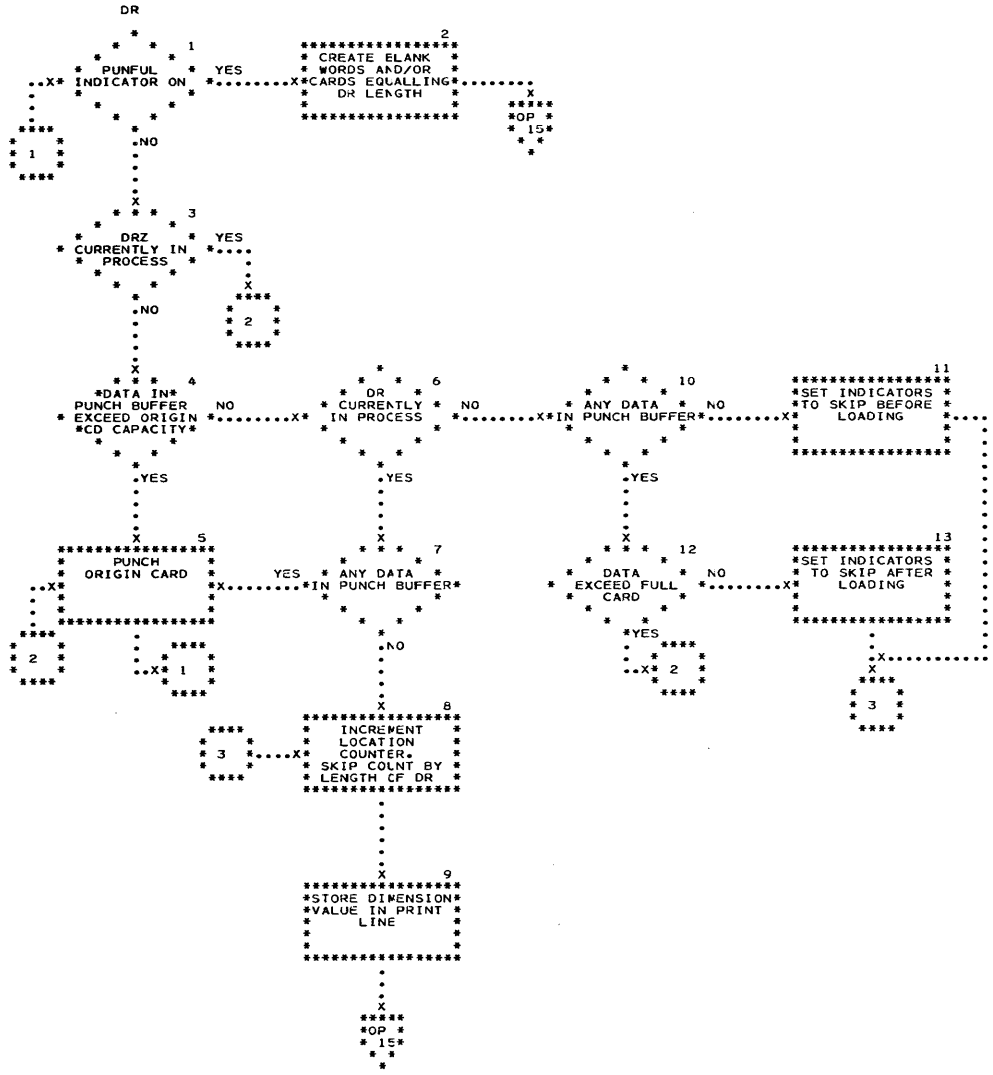

PUNALL

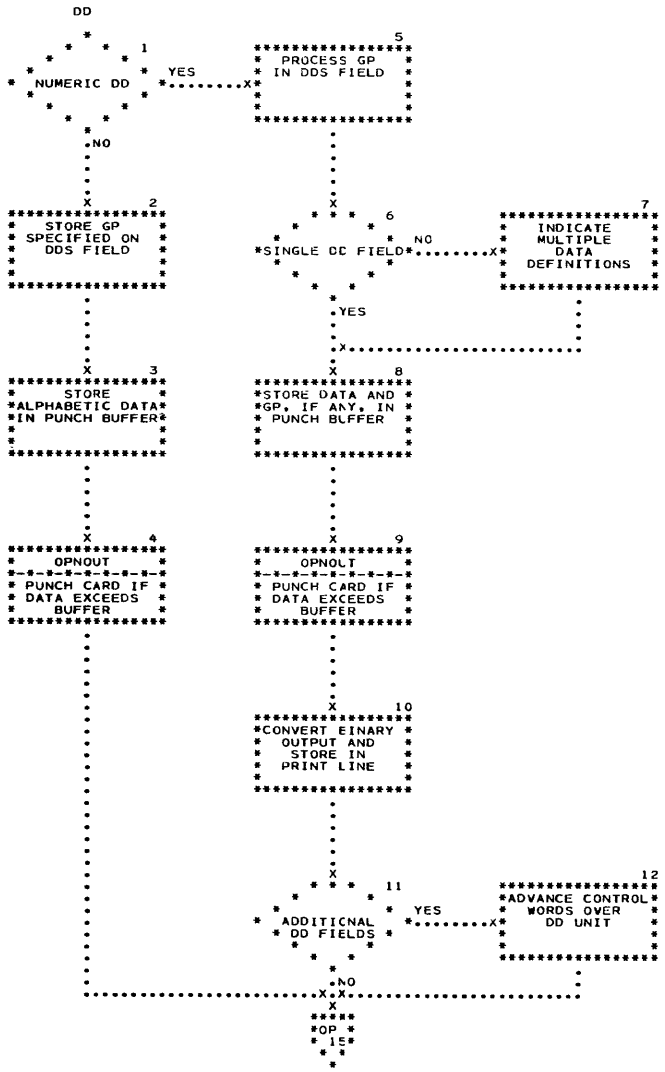



SPNUS




NOSEQ
PUNCH CARD IN EXISTING MODE
TURN ON BIT TO PREVENT PUNCHING SEQUENCE NUMBER
*OP * 14*
RESEQ
PUNCH CARD IN EXISTING MODE
TURN OFF BIT TO PREVENT PUNCHING SEQUENCE NUMBERS
CLEAR SEQUENCE NUMBER FIELD
*OP * 14*
PRNTALL
TURN ON BIT TO ALLOW PRINTING OF ENTIRE SYMBOL TABLE
*OP * 14*
DUPLI
*OP * 14*


DR
1
PUNFUL INDICATOR ON
YES
NO
2
CREATE BLANK WORDS AND/OR CARDS EQUALLING DR LENGTH
OP 15
3
DRZ CURRENTLY IN PROCESS
YES
NO
2
4
DATA IN PUNCH BUFFER EXCEED ORIGIN CD CAPACITY
NO
YES
6
DR CURRENTLY IN PROCESS
NO
YES
10
ANY DATA IN PUNCH BUFFER
NO
YES
11
SET INDICATORS TO SKIP BEFORE LOADING
5
PUNCH ORIGIN CARD
YES
7
ANY DATA IN PUNCH BUFFER
NO
12
DATA EXCEED FULL CARD
NO
YES
2
13
SET INDICATORS TO SKIP AFTER LOADING
3
1
3
INCREMENT LOCATION COUNTER. SKIP COUNT BY LENGTH OF DR
8
STORE DIMENSION VALUE IN PRINT LINE
9
OP 15


DD
1
NUMERIC DD
YES
NO
2
STORE GP SPECIFIED ON DDS FIELD
3
STORE ALPHABETIC DATA IN PUNCH BUFFER
4
OPNOUT
PUNCH CARD IF DATA EXCEEDS BUFFER
5
PROCESS GP IN DDS FIELD
6
SINGLE DD FIELD
NO
YES
7
INDICATE MULTIPLE DATA DEFINITIONS
8
STORE DATA AND GP, IF ANY, IN PUNCH BUFFER
9
OPNOLT
PUNCH CARD IF DATA EXCEEDS BUFFER
10
CONVERT BINARY OUTPUT AND STORE IN PRINT LINE
11
ADDITIONAL DD FIELDS
YES
NO
12
ADVANCE CONTROL WORDS OVER DD UNIT
OP 15

LINKAGE

The calling sequence is: SIC, Outgz; B, Outgo*.

DETAILS

During Pass 2b, as each unit is completed, control is given to OUTPUT to create the listing print line and store the binary output into the punch buffer, punching a card whenever necessary. OUTPUT uses the subroutines, OPLIST and OPPUN, for listing and punching. The mechanics of page headings, numbering and overflow are done by OEDIT. At the end of the entire assembly, the following subroutines are used:

NQBTP: if the PUNSYM or PUNALL pseudo-ops have been specified.

NUNDSY: prints lists of undefined, multiply and circularly defined symbols, plus the entire symbol table if PRNTALL was specified.

ERRPRT: prints the list of error messages.

FLOW CHART DESCRIPTION

Box 6

The subroutine OEDIT is used to print the page heading and program boundaries for the assembly.

Box 2

The error flags for undefined, multiply and pseudo-defined symbols and contagious errors are printed here.

Boxes 15, 16

IOD cards specified by the programmer are punched, as well as the following:

Absolute decks: TYPE, GO
LIM,_____,_____

Relocatable decks: TYPE, GO
LIM,_____,_____

Fortran decks: TYPE, GO, FORTRAN

Boxes 20, 24, 25

The action taken for the various pseudo-ops, is explained in separate flow charts.

* This subroutine is referred to in the text as OUTPUT not as OUTGO.

Box 17

A separate flow chart is provided for DD's.

Boxes 9, 10, 11, 18, 22, 26

The binary output is stored in the buffer, after which OUTPUT determines whether there is enough data to punch a card in the existing punch mode. OUTPUT uses the routine OPPUN to perform the punch function.

Box 12

The binary output of the instruction is picked up from word 5 of the instruction unit, converted and stored in the print image.

Boxes 14, 19, 23

If there is a name associated with the instruction, OUTPUT uses NAMEIN to get it from the name file, and stores it in the print image. Depending on the length of the name, this may require an extra line.

Box 5

The symbolic statement also may require more than one line for the listing.

Box 28

OEDIT inserts the carriage control character and line number in the print image, as well as headings where a new page is required.

Boxes 29, 37

If there have been any errors associated with the unit in process, the page and line number are stored in the error entry by the subroutine ERRNUM.

Boxes 31, 32

When the END unit of the assembly is processed, a final check is made to ascertain that the name file has been exhausted and therefore correctly matched with the instructions.

Boxes 33, 38

The PUNSYM subroutine constructs card images and punches cards for each symbol requested, or all

symbols if PUNALL was specified. The SYN card format is included in the NQBTP subroutine description.

Box 34

NUNDSY scans the symbol table and prints lists of undefined, multiply and circularly defined symbols, plus a list of the entire symbol table with the location at which each symbol is defined, if PRNTALL was specified in the assembly.

Box 35

ERRPRT sorts the error list into page and line number sequence and lists it via OPLIST, followed if necessary by a message stating the number of errors not listed because of exceeding the error buffer.

Boxes 36, 39

OEDIT now prints a time message and the number of units in the assembly, after which control reverts to MCP for a normal EOJ.

DR PROCESSING IN OUTPUT

Boxes 1, 2

Since no control information can be punched in a PUNFUL card, blank words and/or cards must be punched equivalent to the DR length.

Box 3

If there is a DRZ already waiting when a DR is specified, an origin card must be punched immediately before the DR can be processed.

Boxes 6-8

If a DR is waiting when another DR is specified, and there is data in the buffer, an origin card must be punched immediately; if there is no data, the DR length can be added to the previous DR length.

Boxes 10-13

If no DR is waiting and no data in the buffer, the card indicators are set to "skip before loading." If there is some data, the indicators are set to "skip after loading." If there is more than enough data to fill an origin card, a card is punched before the new DR is processed.

DD PROCESSING IN OUTPUT

Box 1

Numeric DD's are identified by the bit Ziotpt.25 being on in the instruction unit. This was turned on by the MQDD subroutine in Pass 1.

Boxes 2, 3

If a general parenthetical field integer entry (gp) was specified in the dds of an alphabetic DD, it is OR'ed at O6pa1 into the alphabetic data in the instruction unit, which is then picked up and stored in the punch buffer.

Box 4

OPNOUT is used to determine whether or not a card should be punched, based on the type of card being punched.

Box 5

A gp specified in the dds field of a DD must be OR'ed into each unit of a multiple unit DD; it is therefore stored in a temporary area to be used for each unit.

Boxes 6, 7

If a multiple unit DD is in process, Zinfld in the instruction unit will show the number of units, and OUTPUT turns on the indicator bit Oenm so that after the first field is printed, OUTPUT can return to process the others instead of returning to the main flow of Pass 2b.

Box 8

All the gp fields are OR'ed into the data itself, which is then stored in the punch buffer, updating the bit count accordingly. For signed data the sign byte is stored separately in the punch buffer.

Box 9

OPNOUT determines whether to punch a card, and if so, fills in the control data such as ID sequence, card type code, origin, etc. .

Box 10

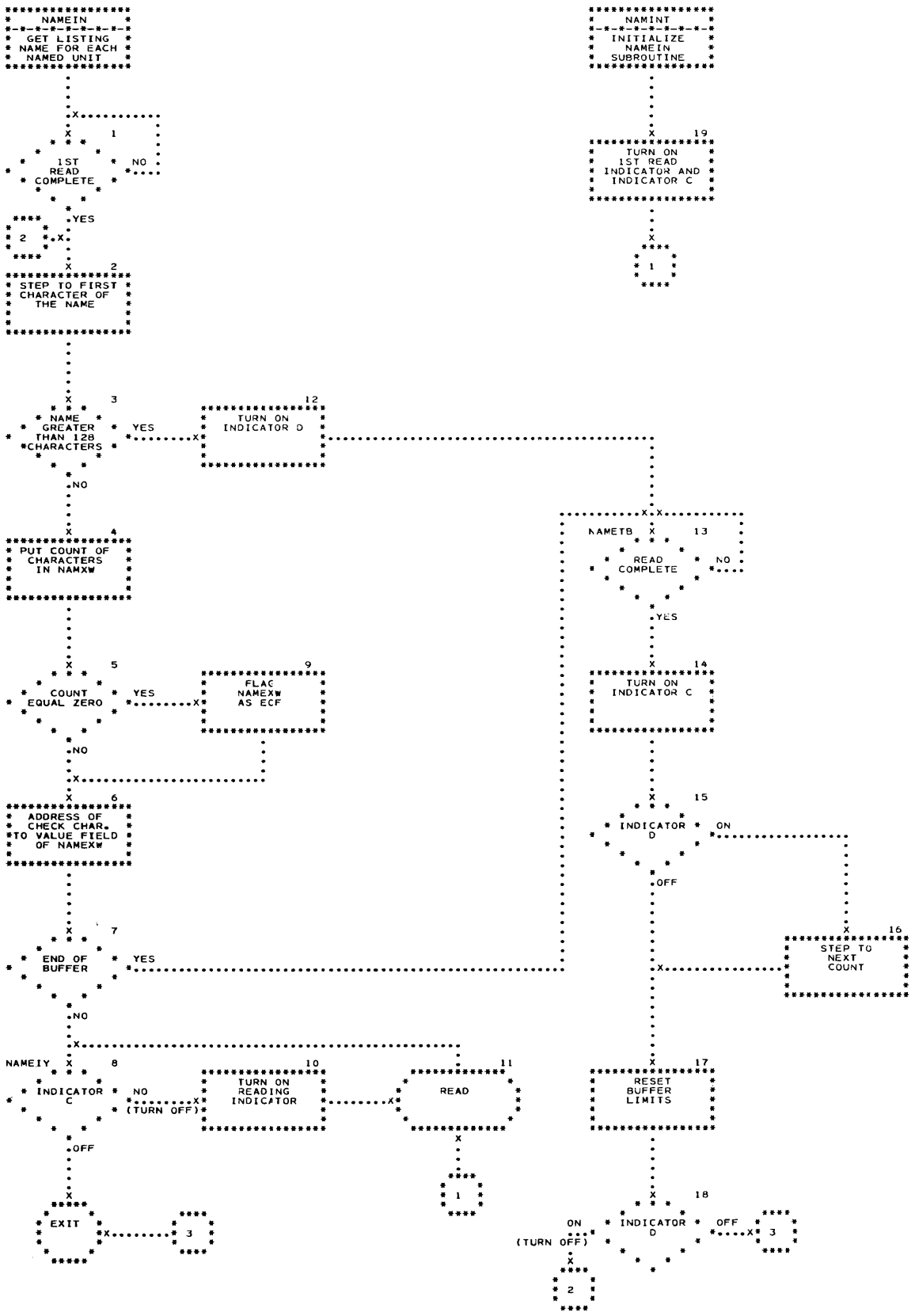
The DD output is converted and stored in the print line according to the type of DD. DD's are not printed in octal-hex format, but in the following:

Floating point: Exponent, fraction, sign byte.
Fixed point: Sign byte separate for signed data.

Boxes 11, 12

If the bit Oenm is on, there are additional fields in the DD unit, and the index word pointing to the beginning of the D field must be advanced to the next field by a standard 120 bits plus any extra bits for Fn, special exponent or gp depending on the format of the existing field.

After printing the location counter, binary output and symbolic statement, if the multiple unit DD bit (Oenm) is still on, OUTPUT will print the new location counter setting and return to box 2 to process the next unit.


NAMEIN
GET LISTING NAME FOR EACH NAMED UNIT
1ST READ COMPLETE
NO
YES
2
STEP TO FIRST CHARACTER OF THE NAME
3
NAME GREATER THAN 128 CHARACTERS
YES
12
TURN ON INDICATOR D
NO
4
PUT COUNT OF CHARACTERS IN NAMXW
5
COUNT EQUAL ZERO
YES
9
FLAG NAMEXW AS ECF
NO
6
ADDRESS OF CHECK CHAR. TO VALUE FIELD OF NAMEXW
7
END OF BUFFER
YES
NO
NAMEIY
8
INDICATOR C
NO (TURN OFF)
10
TURN ON READING INDICATOR
11
READ
OFF
EXIT
3
1
NAMINT
INITIALIZE NAMEIN SUBROUTINE
19
TURN ON 1ST READ INDICATOR AND INDICATOR C
1
NAMETB
13
READ COMPLETE
NO
YES
14
TURN ON INDICATOR C
15
INDICATOR D
ON
OFF
16
STEP TO NEXT COUNT
17
RESET BUFFER LIMITS
18
INDICATOR D
OFF
ON (TURN OFF)
2
3

LINKAGE

On Entry

1. Initially the calling sequence is:
SIC, Nameiz; B, Namint.
2. Subsequently the calling sequence is:
SIC, Nameiz; B, NAMEIN.

On Exit

1. This subroutine reads the name file from the disk.
2. The address in memory of the name check character is placed in the value field of Namexw.
3. When the file is exhausted, the flag, bit 25 in Namexw, is turned on.

FLOW CHART DESCRIPTION

Boxes 1, 19

During initialization, indicators A, B, and C, are turned on and a read operation is initiated. Each read EOP interrupt fix-up turns off indicators A and B. This box waits for the first read operation to be completed and for indicator A to be set to zero.

Boxes 2-4

The character count is loaded into the accumulator from $2, the latter being stepped to the beginning of the name. The count is checked against 128. If less, the count is placed in the count field of Namexw.

Boxes 5, 9

The count is checked for zero. If zero, the end-of-file has been reached and the flag in Namexw is set on.

Box 6

The address of the name check character is stored in the value field of Namexw.

Boxes 7, 8, 10, 11

The count of the next name is added to the present location in the buffer. If the result exceeds the end of the buffer, another read will be required if in the second buffer. If not, indicator C is tested. If off, return is made. If on, it is reset, indicator B is turned on, and a read initiated.

Boxes 12-14

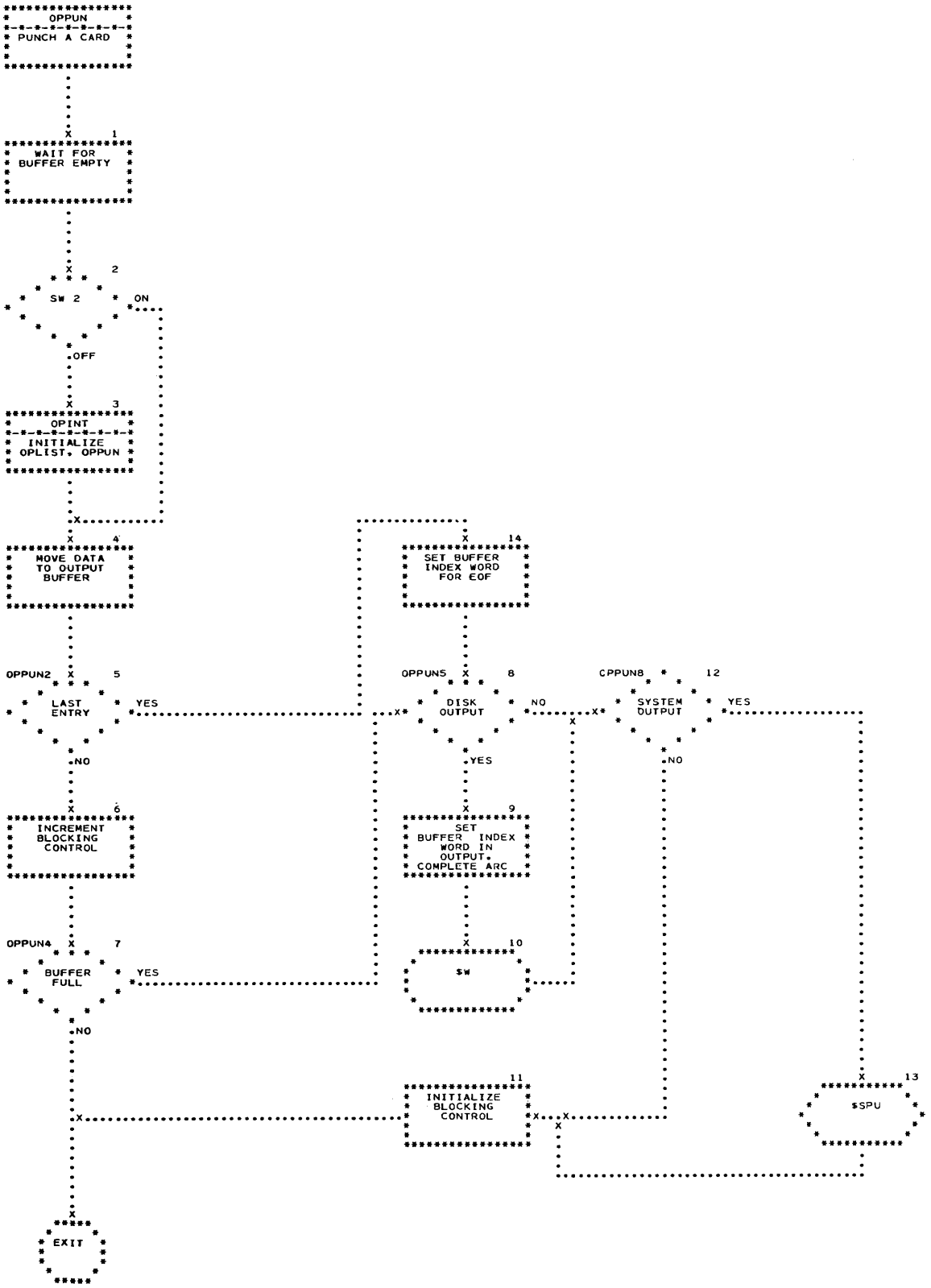
Indicator D is turned on indicating that the end of the second buffer has been reached. The program waits for indicator B to be reset to zero by an EOP interrupt, and indicator C is turned on (indicating a read is required).

Boxes 15-17

If indicator D is on, the locator is reset to the first buffer. The buffer limits are initialized.

Box 18

If indicator D is off, return is made; if on, it is reset, and the next name is obtained.


OPPUN
PUNCH A CARD
1
WAIT FOR BUFFER EMPTY
2
SW 2
ON
OFF
3
OPINT
INITIALIZE OPLIST, OPPUN
4
MOVE DATA TO OUTPUT BUFFER
OPPUN2
5
LAST ENTRY
YES
NO
6
INCREMENT BLOCKING CONTROL
OPPUN4
7
BUFFER FULL
YES
NO
EXIT
OPPUN5
8
DISK OUTPUT
NO
YES
9
SET BUFFER INDEX WORD IN OUTPUT, COMPLETE ARC
10
$W
11
INITIALIZE BLOCKING CONTROL
14
SET BUFFER INDEX WORD FOR EOF
CPPUN8
12
SYSTEM OUTPUT
YES
NO
13
$SPU

LINKAGE

On Entry

1. The usual calling sequence is:
   CNOP
   SIC, Oppunz
   B, OPPUN
   CW, Location of data, number of words.
2. In order to terminate the binary output, bit Orien must be set to one before using the above calling sequence to OPPUN.
3. In order to terminate the binary output when a card is not given to OPPUN, the following calling sequence must be used:
   CNOP
   SIC, Oppunz
   B, Oppunx
   NOP; NOP

On Exit

1. If STRAP II is the last compiler in a chain, the binary output will be written on the System Output tape.
2. If compile and go, the output will also be written on the disk starting in relative arc 0. If STRAP II is other than last in a chain, the binary output will be written on the disk as specified in word 10 of the Communication Region. If the binary is to disk, the location of the first arc where the binary is to be placed is specified by the pre-processor.
3. STRAP II will place suceeding binary output arc numbers in word 0 (bits 0-17) of each output arc. The number of card images in the arc is placed in bits 28-45. Bit 25 on indicates the final arc of binary output.

FLOW CHART DESCRIPTION

Box 1

The program waits for bit Opfull to be reset by the write EOP interrupt, indicating that the binary buffer is ready to accept data.

Boxes 2, 3

Switch 2 is a NOP-Branch switch which is set to a branch by OPLIPU. There is an identical switch on OPLIST. The first time through either OPLIST or OPPUN will cause OPLIPU to initialize both routines.

Box 4

The card image is moved to the binary output buffer under control of the blocking control index word (Opblk), which contains the working location in the binary buffer and the number of available positions in the buffer.

Box 5

Orien is checked to see if the current entry is the last. If so, the buffer will be written regardless of the blocking control count.

Box 6

The blocking control is stepped over the card image in the buffer.

Box 7

The count in Opblk is decremented and checked for zero. If zero, the buffer is full and ready for output. If not zero, Opblk is saved and return is made.

Boxes 8-10

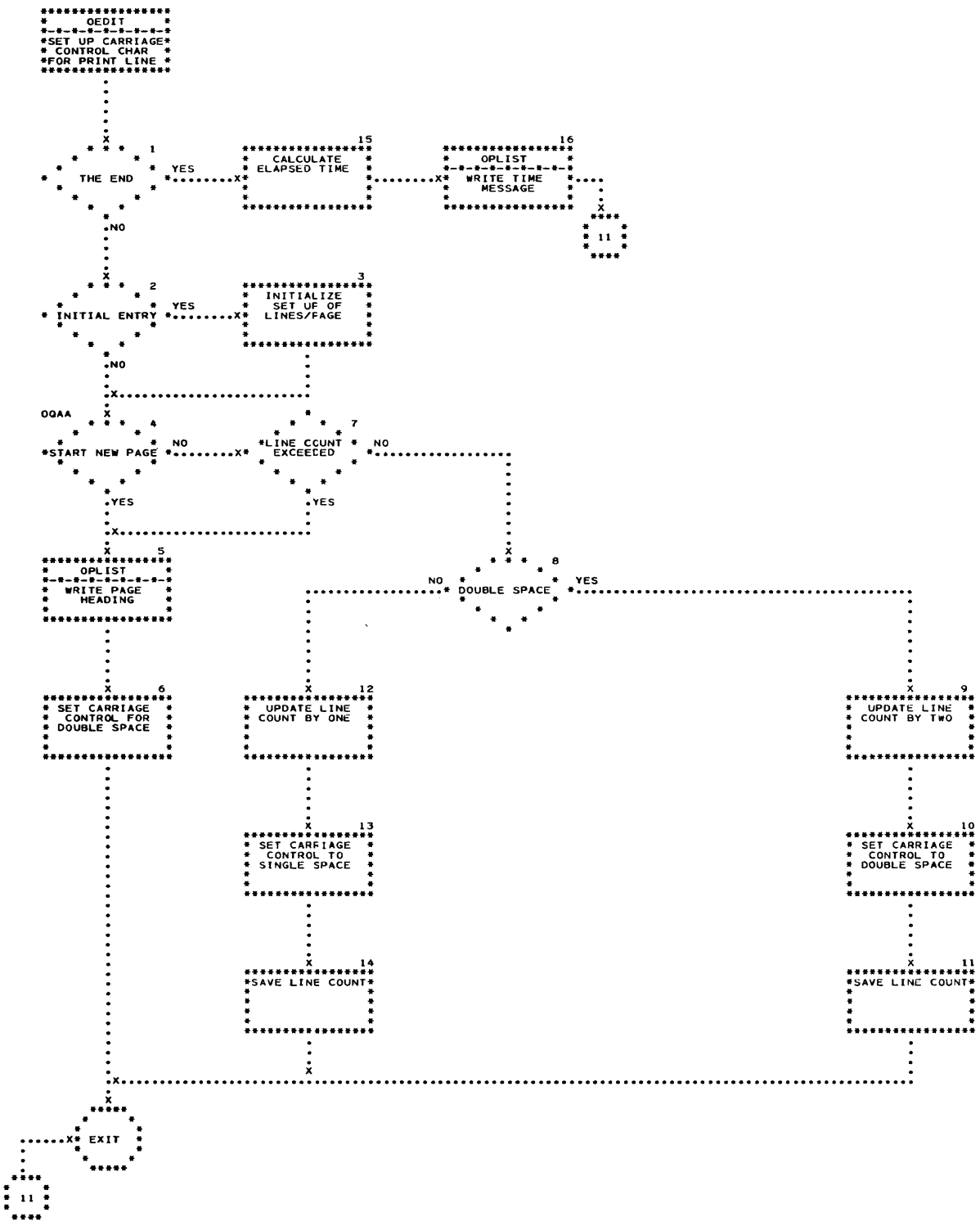
If bit Opdisk has been turned on by OPLIPU, the next available arc for binary output is obtained from Oarcx and placed in the first word of the buffer. Oarcx is stepped to the next available arc for listing or binary output. The number of cards in the buffer is stored in the count field of the first word, and the arc is written on the disk.

Boxes 12, 13

If bit Spool2 has been turned on by OPLIPU, the binary output buffer will be written on the System Output tape.

Box 11

The blocking control is reset, indicating an empty buffer (i.e. starting at word 2, and maximum card count).


QEDIT
SET UP CARRIAGE CONTROL CHAR FOR PRINT LINE
1
THE END
YES
NO
15
CALCULATE ELAPSED TIME
16
QPLIST WRITE TIME MESSAGE
11
2
INITIAL ENTRY
YES
NO
3
INITIALIZE SET UP OF LINES/PAGE
QQAA
4
START NEW PAGE
NO
YES
7
LINE COUNT EXCEEDED
NO
YES
5
QPLIST WRITE PAGE HEADING
8
DOUBLE SPACE
NO
YES
6
SET CARRIAGE CONTROL FOR DOUBLE SPACE
12
UPDATE LINE COUNT BY ONE
9
UPDATE LINE COUNT BY TWO
13
SET CARRIAGE CONTROL TO SINGLE SPACE
10
SET CARRIAGE CONTROL TO DOUBLE SPACE
14
SAVE LINE COUNT
11
SAVE LINE COUNT
EXIT
11

LINKAGE

OEDIT is entered by the calling sequence --
SIC, Oedita; B, OEDIT -- with certain indicators set according to the type of entry:

| | | |
|---|---|---|
| Oqend | = | 1 if the end time message is to be printed. |
| Oqinit | = | 1 if initial entry. |
| Onpge | = | 1 if new page is to be started (e.g., for a SKIP). |
| Odbsp | = | 1 if double spacing is in effect. |

FLOW CHART DESCRIPTION

Boxes 1, 2, 4, 8

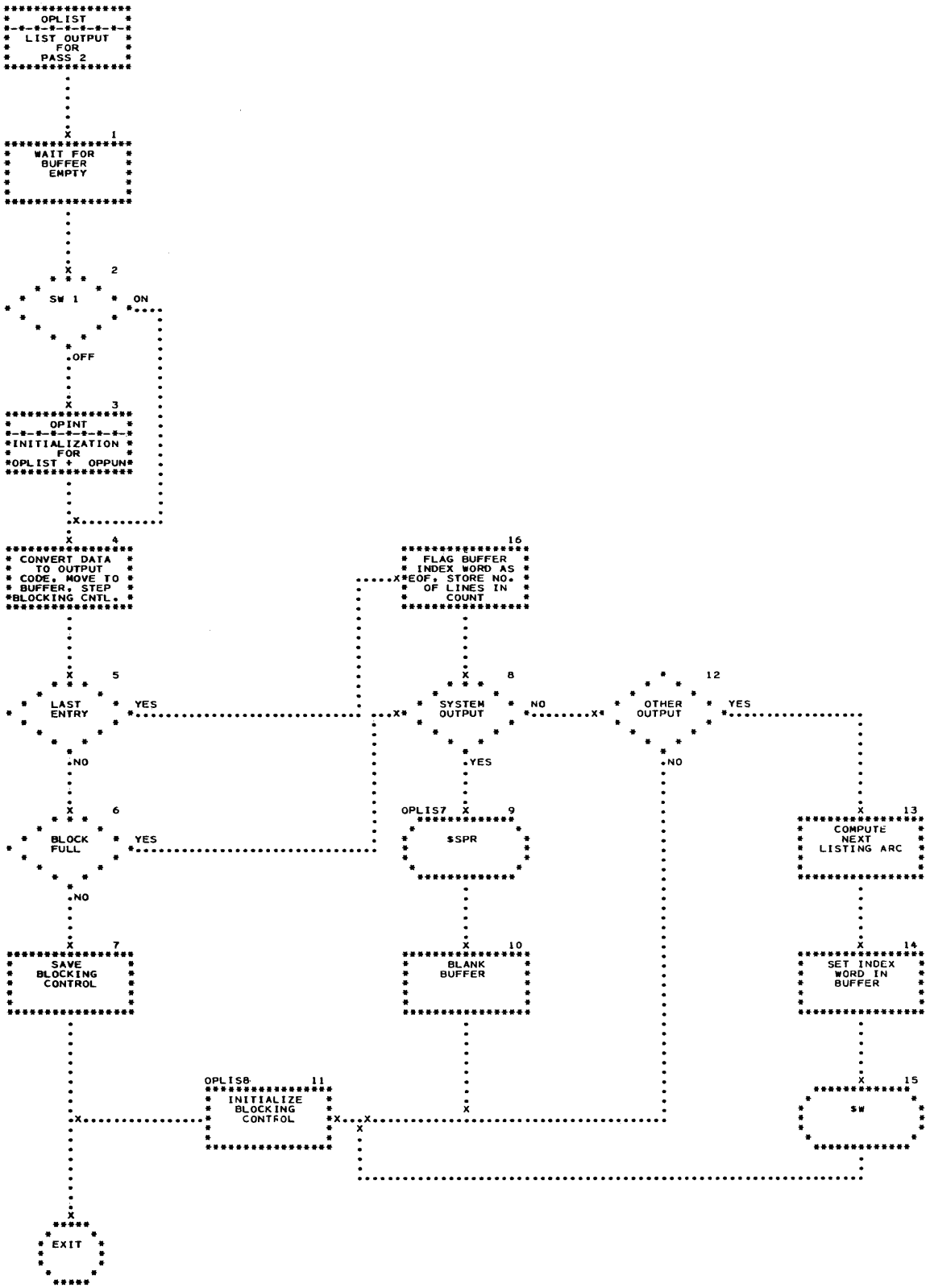
These tests are determined by the indicator(s) set by OUTPUT.

Boxes 7-14

Note that the line count maintained by OEDIT is for the physical number of lines allowed per page. (The line count maintained by OUTPUT is for the purpose of numbering the lines of the instructions.) The number of lines allowed per page is set up in the Communication Region of STRAP II.

Box 5

The page heading includes the initial time clock reading, the card identification from columns 73-80 of the card, the identification from the most recent PRNID pseudo-op, the page number, the setting of location counter (within a word) at the start of this page, and the title headings for the columns on the listing.


OPLIST
LIST OUTPUT FOR PASS 2
1
WAIT FOR BUFFER EMPTY
2
SW 1
ON
OFF
3
OPINT
INITIALIZATION FOR OPLIST + OPPUN
4
CONVERT DATA TO OUTPUT CODE, MOVE TO BUFFER, STEP BLOCKING CNTL.
5
LAST ENTRY
YES
NO
6
BLOCK FULL
YES
NO
7
SAVE BLOCKING CONTROL
8
SYSTEM OUTPUT
NO
YES
9
OPLIS7
$SPR
10
BLANK BUFFER
11
OPLIS8
INITIALIZE BLOCKING CONTROL
12
OTHER OUTPUT
YES
NO
13
COMPUTE NEXT LISTING ARC
14
SET INDEX WORD IN BUFFER
15
SW
16
FLAG BUFFER INDEX WORD AS EOF, STORE NO. OF LINES IN COUNT
EXIT

LINKAGE

On Entry

1. The calling sequence is:
   CNOP
   SIC, Oplisz
   B, OPLIST
   CW, Location of data, number of words.
2. In order to terminate the listing, bit Oprien must be set to one before branching to OPLIST.

On Exit

1. If STRAP II is the last compiler in a chain, the listing will be written on the System Output ($SPR); if not last in a chain, the listing will be written on either the disk or System Output as specified in word 9 of the Communication Region (COMREC). If the listing is to disk, the location of the first arc where the listing is to be placed is specified by the pre-processor.
2. STRAP II will place succeeding listing arc numbers in word 0 (bits 0-17) of each listing arc. The number of lines of listing in the arc is placed in bits 28-45. Bit 25 on indicates the final arc of the listing.

FLOW CHART DESCRIPTION

Box 1

The program waits for bit Olfull to be reset by the write EOP interrupt, indicating that the listing buffer is ready to accept data.

Boxes 2, 3

Switch 1 is a NOP-Branch switch which is set to a branch by OPLIPU. There is an identical switch in OPPUN. The first time thru either OPLIST or OPPUN will cause OPLIPU to initialize both routines.

Box 4

The data is moved to the listing buffer and converted to the code specified by COMREC, or in the case of System Output, to A8 code. The blocking control index (Olblk), which contains the working location in the listing buffer, and the number of available positions in the buffer, is stepped over the line image.

Box 5

Oprien is checked to see if the current entry is the last. If yes, the buffer will be written regardless of the blocking control count.

Boxes 6, 7

The count in Olblk is decremented and checked for zero. If so, the buffer is full and ready for output. If not zero, the blocking control is saved, and return is made.

Boxes 8-10

If bit Spool1 has been turned on by OPLIPU, the listing will be written on the System Output; the buffer will be blanked.

Boxes 12-15

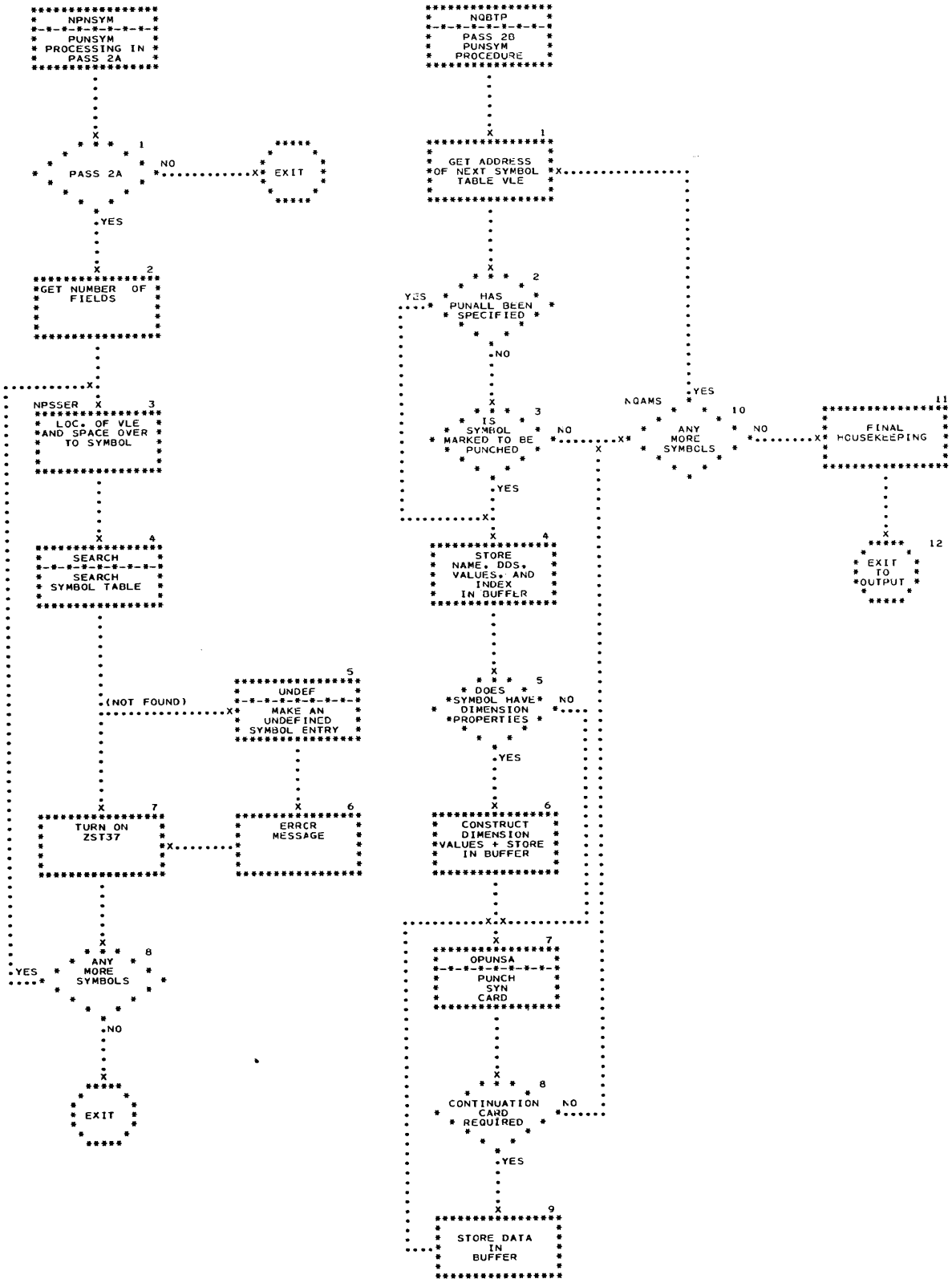
Bit Oldisk will have been set to one if the listing is to disk. If on, the next available arc for listing is obtained from Oarcx and placed in the first word of the buffer. Oarcx is stepped to the next available arc for listing or binary output. The number of lines of listing is stored in the count field of the first word, and the arc is written on the disk.

Box 11

The blocking control is reset, indicating an empty buffer (i.e., starting at word 2, and maximum line count).

Box 16

Bit 25 in word 0 of the listing buffer is set to one, indicating EOF. The count in the blocking control is subtracted from one less than the maximum number of lines that can be placed in the buffer. The result is stored in the count field of word 0.


NPNSYM
PUNSYM
PROCESSING IN
PASS 2A
1
PASS 2A
NO
EXIT
YES
2
GET NUMBER OF
FIELDS
NPSSER
3
LOC. OF VLE
AND SPACE OVER
TO SYMBOL
4
SEARCH
SEARCH
SYMBOL TABLE
(NOT FOUND)
5
UNDEF
MAKE AN
UNDEFINED
SYMBOL ENTRY
6
ERROR
MESSAGE
7
TURN ON
ZST37
8
ANY
MORE
SYMBOLS
YES
NO
EXIT
NQBTP
PASS 2B
PUNSYM
PROCEDURE
1
GET ADDRESS
OF NEXT SYMBOL
TABLE VLE
2
HAS
PUNALL BEEN
SPECIFIED
YES
NO
3
IS
SYMBOL
MARKED TO BE
PUNCHED
NO
YES
4
STORE
NAME, DDS,
VALUES, AND
INDEX
IN BUFFER
5
DOES
SYMBOL HAVE
DIMENSION
PROPERTIES
NO
YES
6
CONSTRUCT
DIMENSION
VALUES + STORE
IN BUFFER
7
OPUNSA
PUNCH
SYN
CARD
8
CONTINUATION
CARD
REQUIRED
NO
YES
9
STORE DATA
IN
BUFFER
NQAMS
YES
10
ANY
MORE
SYMBOLS
NO
11
FINAL
HOUSEKEEPING
12
EXIT
TO
OUTPUT

This subroutine is the PUNSYM procedure in Pass 2b.

LINKAGE

After all binary cards have been punched, OUTPUT branches to the NQBTP (Branch To PUNSYM) only if the switch at Opsym is a Branch. This switch is converted to a Branch by the normal processing of either PUNSYM or PUNALL pseudo-ops by OUTPUT, and is restored to a NOP by the NQBTP subroutine. NQBTP branches directly back to OUTPUT.

DETAILS

The pseudo-op, PUNSYM, initiates the punching of SYN cards for only those symbols specified by the pseudo-op. PUNALL requests punching SYN cards for the entire symbol table. For PUNALL a switch at Nqpall, in addition to the switch at Opsym, must be converted to a Branch by OUTPUT. Both will be restored to NOP's by NQBTP.

The SYN cards punched by STRAP II for symbols requested by the program are in the following format:

| | | |
|---|---|---|
| 2 - 9 | Name | |
| 10 - 12 | Operation Code SYN | |
| 13 - 22 | (Mode, FL, BS), | |
| 23 - 38 | (8) $\pm$ xxxxxx. xx | Bit Value |
| 42 - 53 | (8) $\pm$ xxxxxxxx | Integer Value |
| 56 - 60 | ($xx) | Index Value |
| 61 - 72 | , (Dim.$_1$, Dim.$_2$, . . . , Dim.$_x$) | |
| 73 - 80 | Identification | |

Continuation cards (if required for name, dimension reference, or both):

| | |
|---|---|
| 1 | Asterisk |
| 2 - 9 | Name |
| 10 - xx | Dimension reference |

FLOW CHART DESCRIPTION

Box 1

The address of the first symbol table entry is picked up from the control block (Tsymb + Azfw) and spaced over to the vle entry.

Boxes 2, 3

If the pseudo-op, PUNALL, has been specified in the program, PUNSYM bypasses testing NQBTP to determine those symbols marked to be punched, and punches the entire symbol table. N. B.: STRAP does not punch cards for undefined symbol table entries.

Boxes 10-12

If there are more symbol table entries, NQBTP spaces over to the next vle entry; if not, it restores the original buffer and switch settings, and branches back to Pass 2b.

Box 4

Eight characters of the symbol name are stored in the buffer, and if more characters remain, a switch is set to indicate the necessity for a continuation card. The dds, bit and integer values, and index value are obtained from the symbol table entry and stored in the buffer.

Boxes 5, 6

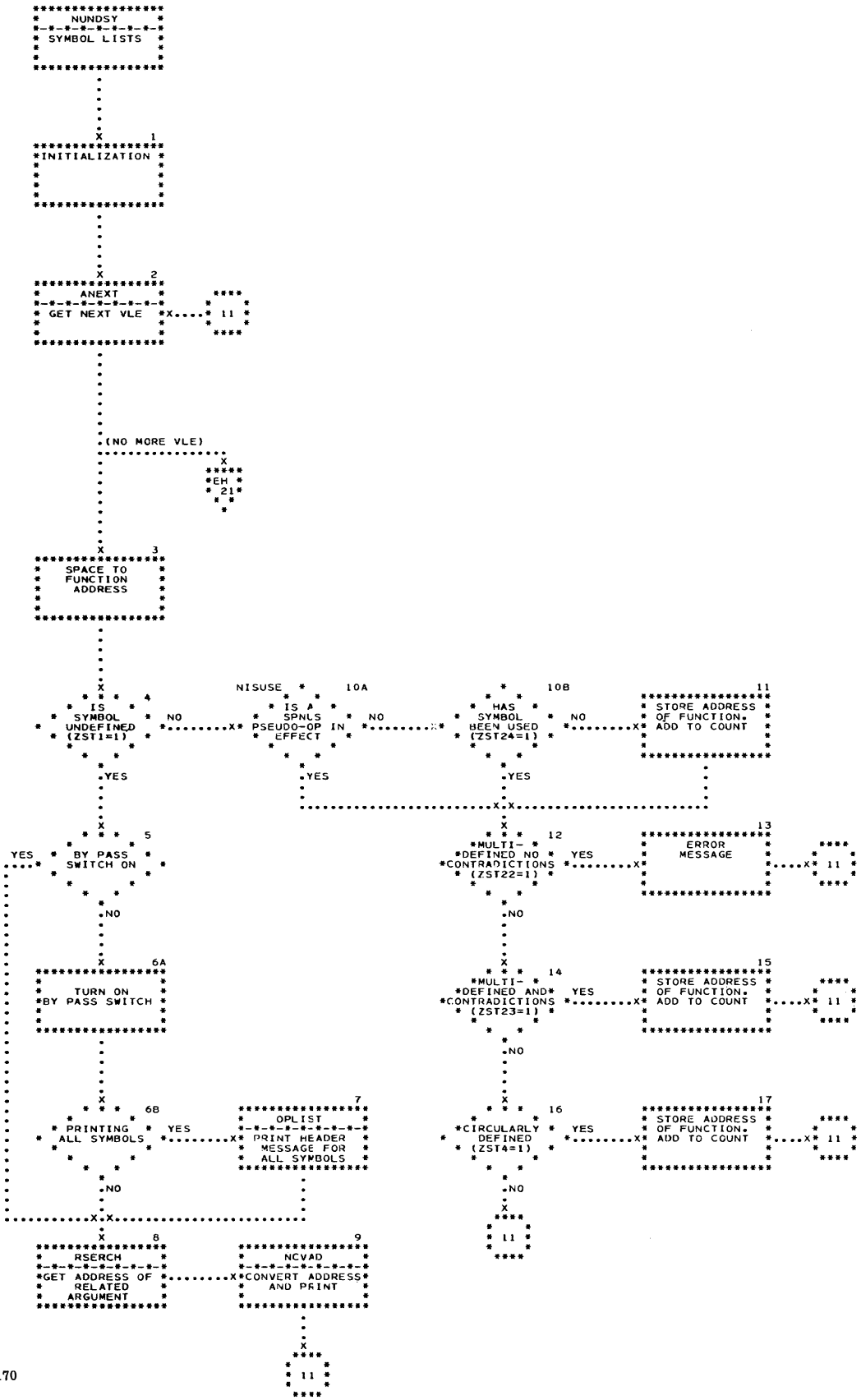
If dimension properties are attached to the symbol, they must be recalculated by NQBTP. Since the original dimensions have already been multiplied to produce the final product of the dimensions, NQBTP divides the last value by the penultimate value, etc., to arrive at the original values.

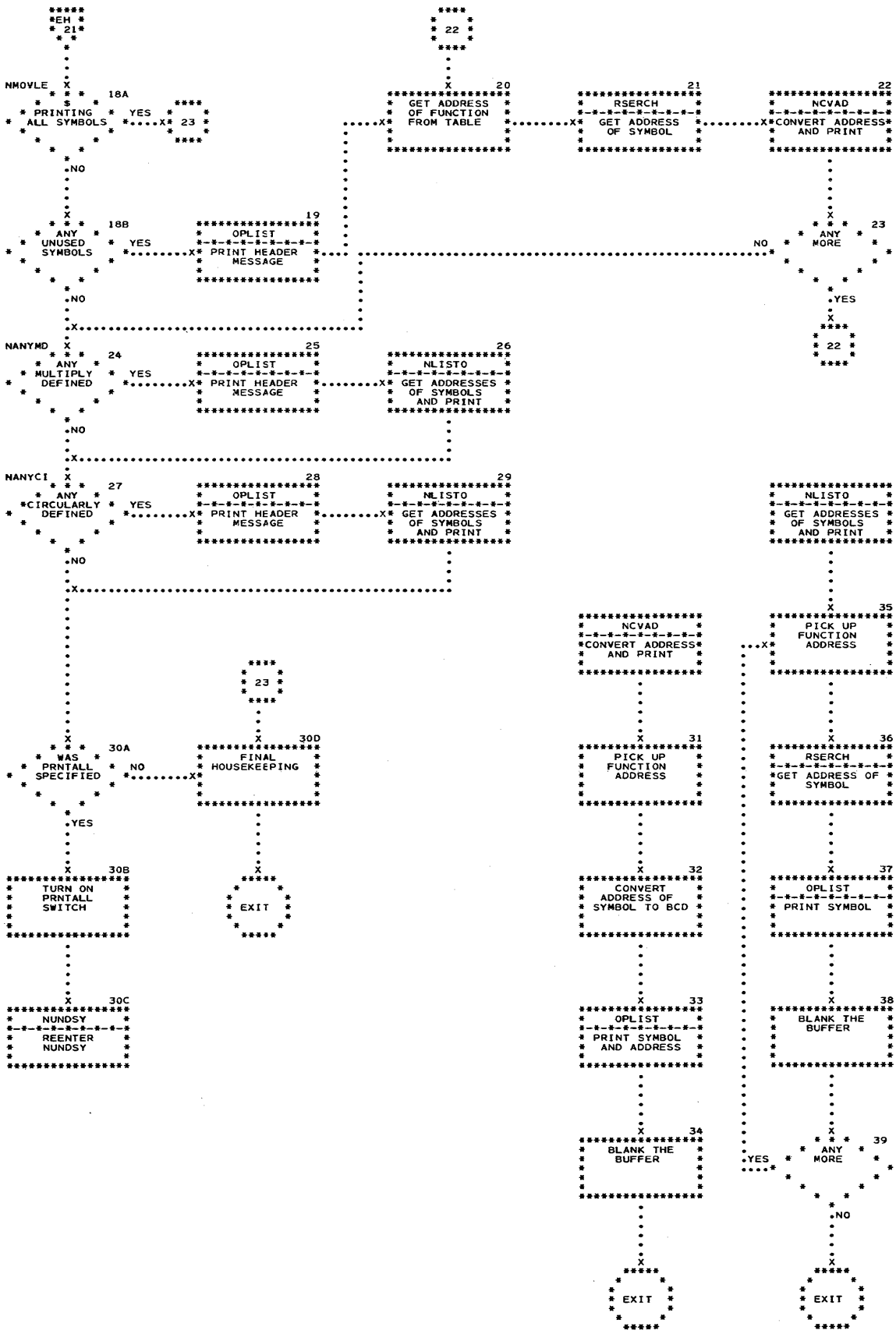
Box 7

Opunsa sets up the control word and branches to OPPUN for the actual punching

Boxes 8, 9

If one or more continuation cards are required because of a name longer than 8 characters and/or an extra number of dimensions, the card image is constructed and NQBTP branches back to box 7.


NUNDSY
SYMBOL LISTS
1
INITIALIZATION
2
ANEXT
GET NEXT VLE
11
(NO MORE VLE)
21
EH
3
SPACE TO FUNCTION ADDRESS
4
IS SYMBOL UNDEFINED (ZST1=1)
NO
YES
NISUSE
10A
IS A SPNUS PSEUDO-OP IN EFFECT
NO
YES
10B
HAS SYMBOL BEEN USED (ZST24=1)
NO
YES
11
STORE ADDRESS OF FUNCTION. ADD TO COUNT
5
BY PASS SWITCH ON
YES
NO
12
MULTI-DEFINED NO CONTRADICTIONS (ZST22=1)
NO
YES
13
ERROR MESSAGE
6A
TURN ON BY PASS SWITCH
14
MULTI-DEFINED AND CONTRADICTIONS (ZST23=1)
NO
YES
15
STORE ADDRESS OF FUNCTION. ADD TO COUNT
6B
PRINTING ALL SYMBOLS
NO
YES
7
OPLIST PRINT HEADER MESSAGE FOR ALL SYMBOLS
16
CIRCULARLY DEFINED (ZST4=1)
NO
YES
17
STORE ADDRESS OF FUNCTION. ADD TO COUNT
8
RSERCH GET ADDRESS OF RELATED ARGUMENT
9
NCVAD CONVERT ADDRESS AND PRINT
11


EH 21
22
NMOVLE
18A
PRINTING ALL SYMBOLS
YES
23
NO
20
GET ADDRESS OF FUNCTION FROM TABLE
21
RSERCH
GET ADDRESS OF SYMBOL
22
NCVAD
CONVERT ADDRESS AND PRINT
18B
ANY UNUSED SYMBOLS
YES
19
OPLIST
PRINT HEADER MESSAGE
NO
23
ANY MORE
YES
22
NANYMD
24
ANY MULTIPLY DEFINED
YES
25
OPLIST
PRINT HEADER MESSAGE
26
NLISTO
GET ADDRESSES OF SYMBOLS AND PRINT
NO
NANYCI
27
ANY CIRCULARLY DEFINED
YES
28
OPLIST
PRINT HEADER MESSAGE
29
NLISTO
GET ADDRESSES OF SYMBOLS AND PRINT
NO
NCVAD
CONVERT ADDRESS AND PRINT
35
PICK UP FUNCTION ADDRESS
23
30A
WAS PRNTALL SPECIFIED
NO
30D
FINAL HOUSEKEEPING
31
PICK UP FUNCTION ADDRESS
36
RSERCH
GET ADDRESS OF SYMBOL
YES
30B
TURN ON PRNTALL SWITCH
EXIT
32
CONVERT ADDRESS OF SYMBOL TO BCD
37
OPLIST
PRINT SYMBOL
30C
NUNDSY
REENTER NUNDSY
33
OPLIST
PRINT SYMBOL AND ADDRESS
38
BLANK THE BUFFER
34
BLANK THE BUFFER
YES
39
ANY MORE
NO
EXIT
EXIT

LINKAGE

OUTPUT branches directly to NUNDSY.

DETAILS

For symbols multiply-defined without contradiction, an error message is executed. Undefined symbols and their addresses are given immediately to the specified output medium, and the locations of the other three types of symbols in error are temporarily stacked, so that only one iteration of the symbol table is necessary. After the complete iteration, the unused, multiply (with contradictions) and circularly defined symbols, if any have been detected, are given to the output medium. For unused symbols, the address is also printed. Only the first 50 characters of the symbol will be printed. The SPNUS pseudo-op suppresses the listing of the unused symbol list.

## ERROR DETECTION AND POST MORTEM DUMPING

As an aid in the original debugging of STRAP II and as a subsequent check on the operation of the computer, there has been provided in the coding of the assembly program tests of various parts of the assembly program's logic in terms of the current instruction being assembled.

In certain areas of STRAP II when specific information should be in the unit according to the instruction being assembled and when specific information should be in the symbol table entry according to the symbol on the instruction being assembled, STRAP inspects the contents of the unit or of the symbol table entry for the specific piece of information. An unsuccessful inspection (i.e., information is not available that should be) causes STRAP to give both an error message containing the location of the test just made and a dump. The dump is optional. If the logic test is made during Pass 1, the dump, if taken, will be of the current unit. If the logic test is made during UITER or Pass 2, the dump, if taken, will be of the vle portion of the current symbol table entry.

Whether or not there is this dump is determined by an indicator in the STRAP II program which STRAP II arbitrarily has defined in the not-to-dump status.

MACHER (MACHine ERror) is the subroutine to handle this error action. The calling sequence to MACHER is SIC, Machez; B, MACHER. The location to be inserted in the message is obtained from the location, Machez. When used by Pass 1, MACHER returns control to the wind-up procedure in MAIN, not to the user routine as would be expected by the calling sequence. Furthermore, when UITER uses MACHER, MACHER marks the vle as completed.

Another checking feature in the assembly program occurs during the matching of files in the Pass 2b phase. To guard against errors while matching names with the units, each name from the name file has associated with it a digit from 0-9. This digit is inserted both in the name record and in the corresponding unit record by Pass 1. The OUTPUT subroutine checks that the files agree.

## DETERMINATION OF PHASE, UNIT, AND SYMBOL ENTRY BEING PROCESSED

In the area of the STRAP program that is particularly reserved for control information common to all phases, there are both a set of indicators whose status indicates the phase of STRAP in process and a set of counters whose contents describe how far the assembly is in process in a particular phase.

The indicators and the meaning of their on-setting are:

Fpass1 = 1, when Pass 1 is in process.
Fuiter = 1, when the iteration between Pass 1 and Pass 2a is in process.
when the iteration between Pass 2a and Pass 2b is in process.
Fpass2 = 1, when the iteration between Pass 1 and and Pass 2a is in process.
when Pass 2a is in process.
when Pass 2b is in process.
Fpass3 = 1, when the iteration between Pass 2a and Pass 2b is in process.
when Pass 2b is in process.

(STRAP II writes the current pass indication on the numeric display lights of the console.)

The counters and their significances are:

1. The Unit counter is set to one at the start of each pass, and is increased by one for each new unit processed. This incrementing is done in Pass 1 after the use of INTOUT, in Pass 2a and Pass 2b after the use of INTIN.

2. The Field counter is set to zero at the start of processing each unit and is increased by one as each address field is processed.

3. The Symbol counter during the UITER subroutine holds the address of the function portion of the variable length entry part of the symbol table entry currently being analyzed by UITER; during Pass 2a and Pass 2b, Symbol holds the address of the function portion of the variable length entry part of the symbol table entry, if any, associated with the current unit. Otherwise, Symbol is zero. (During Pass 1 Symbol always holds the value, Msyte, where the variable length entry part of the symbol table entry is built up.

Furthermore, if the unit has a name, the number associated with each name is carried in the Ziname field of the unit and in the Zstcnt field of the function portion of the variable length entry part of the symbol table entry.

## LOCATIONS IN STRAP II CONTAINING INFORMATION USED TO ANALYZE ERROR CONDITIONS

### Work Area

There is an area (commonly referred to as the work area) within the STRAP II program that is reserved for control-type information. This area is divided

into logical as well as physical sections determined by the phase in which the information is used. One section is used both for the information referred to by more than one phase and for the subroutines used in more than one phase. Then there is a section containing reference information for Pass 1, another reserved for both phases of Pass 2 as well as for the iteration phases, and a final section containing the large buffers (e.g., Tsymba-symbol table, Ibufa-buffer for units).

Sections of STRAP II

| Type Area on Disk | Section Designation | Contents |
|---|---|---|
| SCSTRAP0 | Bootstrap | - |
| SCSTRAP1 | 0 | Those subroutines of STRAP II common to Pass 1 and Pass 2. |
| | A0 | Constants and work areas common to Pass 1 and Pass 2.<br>Constants and work areas for the input-output routines.<br>Error messages common to Pass 1 and Pass 2, all control words for serious and non-serious error messages, and all serious error messages for STRAP II. |
| | A1 | Constants and work areas for Pass 1. |
| | 1 | Subroutines unique to Pass 1. |
| SCSTRAP2 | A2 | Constants and work areas unique to Pass 2.<br>All non-serious error messages. |
| | 2 | Subroutines unique to Pass 2. |
| | A3 | Buffers. |

Index Registers

In STRAP II index registers other than $7, $8, and $9 are at the special disposal of specific routines of the assembly. Therefore, these index registers, the ones other than $7, $8, and $9, are not used generally in STRAP II unless their contents are saved on entrance into and restored on exit from the routine using them.

Index Register Assignment in STRAP II

| Index Register | Contents or Use during Pass 1 Phase | Contents or Use during Pass 2 Phase |
|---|---|---|
| 0 | Value field used for calling sequence reads in TABMAN; refill field used during the execution of a RNX instruction. | Value field used for calling sequence reads in TABMAN; refill field used during the execution of a RNX instruction. |
| 1 | - - - - - | - - - - - |
| 2 | Used in the RNX instruction. | Used in the RNX instruction. |
| 3 | A8 character from GETCHA. | Address of the unit being processed. |
| 4 | - - - - - | Absolute portion of the location counter. |
| 5 | Address of the question bits for instruction being processed | Address of the question bits for instruction being processed. |
| 6 | Address of unit being processed. | Symbolic portion of the location counter (i. e., current 3 character STRAP II special symbol). |
| 7 | Used randomly. | Used randomly. |
| 8 | Used randomly. | Used randomly. |
| 9 | Used randomly. | Used randomly. |
| 10 | Used by VALUE. | Used by VALUE. |
| 11 | Used by VALUE. | Used by VALUE. |
| 12 | - - - - - | Used by NAMEIN. |
| 13 | Used by INTOUT. | Used by OUTPUT. |
| 14 | Used by GETCHA. | Used by NAMEIN. |
| 15 | Value field used for subroutine linkages; refill field used to point to the table of exits for maskable interrupts. | Value field used for subroutine linkages; refill field used to point to the table of exits for maskable interrupts. |

LIST OF THE STRAP II NUMBERED ERROR MESSAGES*

| Number | Logic Area Detecting Error Condition | Message |
|---|---|---|
| 1 | MAIN | ILLEGAL OP CODE 1 |
| 2 | MAIN | ILLEGAL SECONDARY OP 2 |
| 3 | MAIN | ENTRY MODE WITH NON DD OPER. 3 |
| 4 | MAIN | MORE THAN ONE SECONDARY OP 4 |
| 5 | MAIN | MORE THAN ONE DDS 5 |
| 6 | NPS2 | PASS 2A AND 2B LOC. CTR. DO NOT AGREE 6 |
| 7 | MACHER | ASSEMBLY ERROR 7 |
| 8 | GETFLD | VLE SYMBOL TABLE EXCEEDED 8 |
| 9 | GETFLD | SYM. BUFF. HAS BEEN EXCEEDED 9 |
| 10 | GETCHA | GETCHA BUFFER FULL 10 |
| 11 | | SPARE 11 |
| 12 | OUTPUT | REACHED FLAG IN NAMEXW PRIOR TO END INSTRUCTION 12 |
| 13 | OUTPUT | FLAG NOT SET IN NAMEXW AT END INSTRUCTION 13 |
| 14 | OUTPUT | NAME CHECK CHARACTERS DO NOT COMPARE 14 |
| 15 | | SPARE 15 |
| 16 | | SPARE 16 |
| 17 | | SPARE 17 |
| 18 | MAIN | SYM TBL ENTRY MADE UNORDERED 18 |
| 19 | ANEXT | SYMBOL TABLE IS INCORRECT 19 |
| 20 | | SPARE 20 |
| 21 | | SPARE 21 |
| 22 | | SPARE 22 |
| 23 | | SPARE 23 |
| 24 | | SPARE 24 |
| 25 | | SPARE 25 |
| 26 | | SPARE 26 |
| 27 | | SPARE 27 |
| 28 | | SPARE 28 |
| 29 | MAIN | IMPROPER 1ST CHAR 29 |
| 30 | MAIN | MORE THAN ONE $ 30 |
| 31 | MAIN | ILLEGAL ENTRY MODE 31 |
| 32 | MAIN | ENTRY MODE NOT CLOSED BY RIGHT PAREN 32 |
| 33 | MAIN | 2NDARY OP NOT CLOSED BY RIGHT PAREN 33 |
| 34 | MAIN | THIS OP SHOULD NOT HAVE DDS 34 |
| 35 | MAIN | DDS NOT CLOSED BY RT. PAREN 35 |
| 36 | MAIN | FIELD LENGTH GREATER THAN 64 36 |
| 37 | MAIN | BYTE SIZE GREATER THAN 8 37 |
| 38 | MAIN | BIT STYLE NO. IN DDS 38 |
| 39 | MAIN | NEG. FL OR BS HAS BEEN COMPLEMENTED 39 |
| 40 | MAIN | EXTRA FIELDS 40 |
| 41 | MAIN | SHOULD HAVE NO NAME 41 |
| 42 | MAIN | STRAP ASSIGNED DDS 42 |
| 43 | MQS | SYN WITHOUT A NAME 43 |
| 44 | MQS | SYN WITHOUT AN ADDRESS FIELD 44 |
| 45 | MQDD | UNATTAINABLE VALUE 45 |

* Error Messages numbered 1-28 can be made to appear on the console typewriter by the use of a Correction card.

| Number | Logic Area Detecting Error Condition | Message |
|---|---|---|
| 46 | MQR | DR OR DD WITHOUT DDS 46 |
| 47 | DNUM | CHAR ILLEGAL IN RADIX SPEC. 47 |
| 48 | DNUM | MORE THAN ONE POINT 48 |
| 49 | CPLTSY | SYMBOL IS TOO LONG 49 |
| 50 | DNUM | MORE THAN ONE E IN NUM. DD 50 |
| 51 | CPLTSY | MORE THAN 1 $ IN SYSTEM SYM. 51 |
| 52 | MDIMRF | MULTIPLE DIMENSIONS NOT IN PAREN 52 |
| 53 | OUTPUT | VALUE ROUNDED TO FULL WORD 53 |
| 54 | MDIMRF | DIMENSION NOT CLOSED BY RIGHT PAREN 54 |
| 55 | MPUNID | NO COMMA AFTER PUNID 55 |
| 56 | CPLTSY | NON-EXISTENT SYSTEM SYMBOL 56 |
| 57 | OUTPUT | PSEUDO LOC. CTR. TOO HIGH 57 |
| 58 | MQTAIL | UNTAIL LEVEL MORE THAN TAIL 58 |
| 59 | MQTAIL | NULL TAIL 59 |
| 60 | MQTAIL | TAIL LEVEL NOT CLOSED BY RIGHT PAREN 60 |
| 61 | MQTAIL | ILLEGAL TAIL LEVEL CHARACTER 61 |
| 62 | MQTAIL | ILLEGAL CHARACTER IN TAIL 62 |
| 63 | CPLTNM | DIGIT INCORRECT FOR RADIX 63 |
| 64 | MQP | ILLEGAL CHARACTER IN PUNSYM 64 |
| 65 | MQDD | MORE THAN 1 RADIX OR PAREN. ENTRY 65 |
| 66 | GETFLD | SYNTAX ERROR 66 |
| 67 | GETFLD | INAPPROPRIATE CHAR. 67 |
| 68 | GETFLD | GP ERROR 68 |
| 69 | VALUE | TRUNCATION IN INDEX VALUE 69 |
| 70 | VALUE | INDEX IN WRONG PLACE, IT IS IGNORED 70 |
| 71 | VALUE | SUBSCRIPT WRITTEN AS BIT 71 |
| 72 | VALUE | CANT SUBSCRIPT CONSTANT, TRY INDEX 72 |
| 73 | VALUE | SUBSCRIPT OR INDEX INCORRECT 73 |
| 74 | VALUE | CANNOT SUBSCRIPT SYMBOL WITH NO DDS 74 |
| 75 | VALUE | 1 SUBS. TOO MANY, LAST USED AS INDEX 75 |
| 76 | VALUE | TOO MANY SUBSCRIPTS, EXTRA IGNORED 76 |
| 77 | VALUE | TOO FEW SUBSCRIPTS, OTHERS TAKEN 0 77 |
| 78 | VALUE | DIVISION BY ZERO, DIVISOR IGNORED 78 |
| 79 | GETCHA | INCORRECT CARD CODE CHAR. 79 |
| 80 | GETCHA | ILLEGAL CHAR. IN FIRST COL. 80 |
| 81 | GETCHA | ILLEGAL CHAR. IN NAME FIELD 81 |
| 82 | GETFLD | (.0) HAS BEEN INTERPRETED AS PAREN INTEGER 82 |
| 83 | MAIN | SYMBOL TOO LONG FOR SPECIFIED TAIL 83 |
| 84 | MQDALF | CC ENTRY MODE WITHOUT BS 12 84 |
| 85 | MQDALF | BS NOT 8 85 |
| 86 | VALUE | ONLY INTEGER VALUES ALLOWED 86 |
| 87 | DECODE | THE FL IS GREATER THAN 64 87 |
| 88 | DECODE | BYTE SIZE GREATER THAN 8 88 |
| 89 | DECODE | BIT TYPE NOT ALLOW. 89 |
| 90 | DECODE | NEG. FIELD HAS BEEN COMPLEMENTED 90 |
| 91 | DECODE | MODE INCONSISTENT WITH OP 91 |
| 92 | DECODE | NO MODE 92 |
| 93 | DECODE | TOO MANY FIELDS 93 |
| 94 | DECODE | ERROR IN GP 94 |
| 95 | DECODE | NEGATIVE GP HAS BEEN COMPLEMENTED 95 |

| Number | Logic Area Detecting Error Condition | Message |
|---|---|---|
| 96 | MDUP | NO FIELDS, STATEMENT IGNORED 96 |
| 97 | DECODE | BIT TYPE UNUSUAL 97 |
| 98 | DECODE | NEGATIVE PARAMETERS ON EXT 98 |
| 99 | DECODE | PARAMETER GREATER THAN 64 99 |
| 100 | NPS2 | ADDR LESS THAN 33. 100 |
| 101 | NSYN | ERR IN DDS 101 |
| 102 | MIOD | ILLEGAL SEQ. OF MCP CARDS 102 |
| 103 | MIOD | IOD CARD SHOULD HAVE NAME 103 |
| 104 | MIOD | I/O TBL OF EXITS ADDR NULL 104 |
| 105 | MIOD | REEL CARD DOESNT NEED NAME 105 |
| 106 | MDUP | MISSING FIELD 106 |
| 107 | MX | EXT NOT FOLLOWED BY CORRECT PARTITION CHAR 107 |
| 108 | GETFLD | GP IS NOT ALLOWED 108 |
| 109 | MAIN | RADIX SPECIFIED AT THE END OF A DD 109 |
| 110 | MQM | BIT STYLE NUMBER USED TO REFER. ERR 110 |
| 111 | MQM | ERR MESSAGE SPECIFIED IS UNKNOWN 111 |
| 112 | MQDDPP | 0 BASE IN DD 112 |
| 113 | MQDDPP | EXP. NOT IN RANGE 113 |
| 114 | INSERT | NEGATIVE FIELD HAS BEEN COMP. 114 |
| 115 | INSERT | INDEX NOT ALLOWED 115 |
| 116 | INSERT | ADDRESS FIELD HAS BEEN TRUNCATED 116 |
| 117 | INSERT | ONLY K FIELD ALLOW. 117 |
| 118 | INSERT | ADDR INCLUDES BITS NOT NORMAL IN OP 118 |
| 119 | INSERT | SLC HAS AN INTEGER 119 |
| 120 | INSERT | BIT TYPE ADDR UNUSUAL HERE 120 |
| 121 | INSERT | BIT ADDRESSED TWICE IN LVS OR INDMK 121 |
| 122 | NEXT | INSTR. NOT ALLOWED IN EXT 122 |
| 123 | INSERT | MORE THAN 1 LOC. CTR. DEP. SYMBOL 123 |
| 124 | NPNSYM | SYM ON PUNSYM NOT IN PROG. 124 |
| 125 | MQDALF | NO TERMINATING CHARACTER 125 |
| 126 | GETCHA | FORCED COMMENT CARD 126 |
| 127 | OUTPUT | PUNID IN PUNFUL SEQUENCE -- STATE. IGNORED 127 |
| 128 | NUNDSY | MULTIPLY ASSUMED 128 |
| 129 | MAIN | OP OR SS CHANGED INTO LEGAL MNEMONICS 129 |
| 130 | DECODE | FIELD LENGTH MORE THAN 24 130 |
| 131 | NUNDSY | MULT. DEFINED SYM WITH NO CONTRADICTION 131 |
| 132 | DECODE | CANNOT EVALUATE DDI 132 |
| 133 | NEXT | INCONSISTENCY IN EXT PARAMETERS 133 |
| 134 | OUTPUT | PSEUDO LOC. COUNTER CHECK 134 |
| 135 | GETCHA | ID SEQUENCE 135 |
| 136 | OUTPUT | RELOCATABLE PSEUDO-OP., NOT IN PUNREL MODE 136 |
| 137 | OUTPUT | COMBLOCK STATEMENT WHEN NOT IN PUNCDC MODE 137 |
| 138 | OUTPUT | ENTER STATEMENT, NOT IN PUNFPC MODE 138 |
| 139 | OUTPUT | NAME LONGER THAN 8 CH. ON COMBLOCK OR ENTER 139 |
| 140 | MAIN | NO FIELDS ON RELOCATABLE STATEMENT 140 |
| 141 | MAIN | RELOCATABLE STATEMENT NEEDS A NAME 141 |
| 142 | MAIN | COMMON NAME UNDEFINED 142 |
| 143 | MIOD | EXIT ADDR IS NULL 143 |
| 144 | MIOD | NO B ON IOD 144 |
| 145 | MDUP | FIELD IS ZERO 145 |

| | | | |
|---|---|---|---|
| 146 | MDUP | REPEAT PSEUDO-OP WITHIN REPEAT BLOCK, STATEMENT IGNORED 146 | 146 |
| 147 | MDUP | PARAMETER EXCEEDS MAXIMUM ALLOWED 147 | |
| 148 | MDUP | REPEAT PSEUDO-OP ILLEGAL HERE, REPEAT TERMINATED 148 | |

LIST OF THE STRAP II I-O ERROR MESSAGES

| Logic Area Detecting Error Condition | Message |
|---|---|
| XERR | UNCORRECTED ERROR READING DISK<br>RELATIVE ARC___number_____<br>STRAP ASSEMBLY TERMINATED |
| XINPUT | COMREC FOR INPUT FILE IN ERROR<br>STRAP ASSEMBLY TERMINATED |
| XERR | UNCORRECTED ERROR WRITING DISK<br>RELATIVE ARC___number_____ |
| INTOUT | THE FOLLOWING INSTRUCTION IS TOO LARGE FOR STRAP BUFFER<br>current contents of the card image buffer |
| OPLIST | COMREC FOR LISTING OUTPUT IN ERROR - LISTING ON SYSTEM OUTPUT |
| OPPUN | COMREC FOR BINARY OUTPUT IN ERROR - BINARY ON SYSTEM OUTPUT |
| OPPUN | PUNFUL, COMPILE AND GO BOTH SPECIFIED |
| OPLIST or OPPUN | INSUFFICIENT DISK STORAGE FOR STRAP OUTPUT<br>STRAP ASSEMBLY TERMINATED |

## INCLUDING A COPROCESSOR SUBROUTINE WITH STRAP II

There can be included with the assembly program a coprocessor subroutine which Pass 1 can enter whenever any one of six conditions occurs during the Pass 1 phase. The conditions are:

1. An illegal STRAP or MCP character in column 1 of the input.(This condition is detected by GETCHA.)
2. An illegal programmer symbol in columns 2-9 of the input--where the first character of a name must be alphabetic; the subsequent characters of the name must be alphameric. (This condition is detected by GETCHA.)
3. A non-IBM card code character punched on the card input. (This condition is detected by XINPUT.)
4. Immediately before the pseudo-op END is processed. (This condition is detected by GETCHA.)
5. An illegal operation code. (This condition is detected by MAIN.)
6. An inappropriate character in an entry mode --where the current entry modes allowed by STRAP II are: Radix, IQS, CC, A, and Fn. (This condition is detected by MAIN.)

### COMMUNICATION SPECIFICATIONS

#### Entry into Coprocessor

In order for Pass 1 to enter the coprocessor subroutine certain information must be preset in STRAP II's common work area section. If this information is not there when the Pass 1 phase detects one of these six conditions, Pass 1 will treat any possible coprocessor condition strictly as an error condition, and will not consult the coprocessor subroutine at all.

#### Transfer Vector

Therefore, if the coprocessor wants control under any of the above conditions, the common work area of STRAP II must be set up as follows:

1. The bit, Hhcom4, must be on to indicate to STRAP II that the coprocessor wants control during Pass 1 whenever a coprocessor condition is detected on the input.
2. The transfer table at Hhcopr must contain the following when Hhcom4 is on:

```
Hhcopr   B, entry A   'where entry A is the
                      entrance into the co-
                      processor subroutine
                      for the condition 1.
         B, entry B      . . .
         B, entry C      . . .
         B, entry D      . . .
         B, entry E      . . .
         B, entry F      . . .
         B, entry G   'where entry G is the
                      entrance into the co-
                      processor subroutine
                      for the 'WANT ANOTHER
                      CARD' option.
```

NOTE: When Hhcom4 is on and the coprocessor does not wish control after Pass 1 detects a coprocessor condition, the corresponding slot in the transfer table for the unwanted condition must be a B, 0($15). If Hhcom4 is off, the STRAP II transfer vector must be in the transfer table.

GETCHA informs and enters the coprocessor through a special calling sequence after having set up $6 and $7 accordingly.

#### Calling Sequence

GETCHA exits to the coprocessor through the following calling sequence:

```
SIC, $15; B, entry X
   B, ----      ' " CONDITION NOT WANTED"
                  return from the coprocessor
   B, ----      '"INSERT CARDS" return from
                  the coprocessor.
   B, ----      '"WANT ANOTHER CARD"
                  return from the coprocessor.
```

The address entry X corresponds to the appropriate entry address in the previous table.

#### Index Register 7

Index register 7 contains information to be used by the coprocessor.

1. VF of $7 contains the location of the card image with the trouble condition.
2. CF of $7 contains the number of bits which STRAP II has processed on the error card image. (VF + CF = position of pertinent character).
3. RF of $7 contains a value corresponding to the code of the card image being given to the coprocessor.

| Value | Character Code Set |
|---|---|
| 1 | CC code |
| 2 | A6 code |
| 3 | A8 |
| 4 | Not used |
| 5 | IQS code |

4. Bits $7.25 and $7.26 have the following significance:

$7.25 = 0 if STRAP II is giving the coprocessor a card image in the IBM card code.
= 1 if STRAP is giving a card image in a code specified by the RF of $7.

$7.26 = 0 under the usual circumstances. 1 if there is no more material to give the coprocessor when a "WANT ANOTHER CARD" request is made by the coprocessor.

Index Register 6

Under certain of the six conditions, a character in A8 (or card) code is positioned in the rightmost 8 (or 12) bits of the value field of index register 3.

1. If STRAP II enters the coprocessor through entry 1, the VF of $3 contains the illegal STRAP or MCP character in A8 code, positioned as mentioned above.
2. If STRAP II enters the coprocessor through entry 2.the VF of $3 is zero .
3. If STRAP II enters the coprocessor through entry 3, the VF of $3 contains the illegal 12-bit card code character.
4. If STRAP II enters the coprocessor through entry 5. the VF of $3 contains the next character following the operation code where the operation code is assumed to be less than nine characters. If the operation code exceeds eight characters, the VF of $3 contains the ninth. In either case the character is in A8 code.
5. If STRAP II enters the coprocessor through entry 6, the VF of $3 contains the inappropriate entry mode character in A8 code.

(An END pseudo-operation should never be included as part of any insert group to STRAP II. If STRAP II detects the END pseudo-op in insert material, control will be given to the coprocessor through the entry 4 only this once. The bit Hhcom5 in the common area of STRAP will be set on for the coprocessor for this situation.)

Options

The options available to the coprocessor are:

1. The coprocessor may not accept the condition and may re-enter STRAP II through the "CONDITION NOT WANTED" return.
2. The coprocessor may accept the condition; the coprocessor may perform both options 2a and 2b, or just option 2b. For the final return to STRAP II under the 2a option, the coprocessor must re-enter STRAP II through the "INSERT CARDS" return after having setup index register 6 with the prescribed information. Before STRAP II performs the option of "WANT ANOTHER CARD" or of "INSERT CARDS" for the illegal card code character condition, it first terminates the present card block if there are any good units in it.

a. The coprocessor may request a card image -- one at a time -- by a re-entry through the "WANT ANOTHER CARD" return.

b. The coprocessor may or may not insert information after the last unit processed by STRAP II, supplying a group of zero to n card images in a memory buffer specified by the coprocessor.

Exit from the Coprocessor

Whenever the coprocessor accepts a condition, the coprocessor must set up index register 7 with the following information before re-entering Pass 1 through the "INSERT CARDS" routine.

Index Register 7

1. VF of $7 contains the location of the insert buffer material.
2. CF of $7 contains the count + 1 of the card images in the insert buffer. (A CF of 1 indicates that no material is to be inserted; a CF of 0 cannot be specified.)
3. RF of $7 contains a value corresponding to the code of the insert material.
4. Bits $7.25, $7.26 and $7.27 have the following significance:

a. $7.25 = 0 if the buffer contains insert material in the A8 code.
= 1 if the buffer contains material in a code specified by the value in the RF of $7.

b. $7.26 = 0 if the coprocessor only requested as many card images through the "WANT ANOTHER CARD" option as it needed.
= 1 if the coprocessor requested one more card image than it actually needed through this option.

c. $7.27 = 0 if a continuation card may follow the actual insert material of the coprocessor.
= 1 if a continuation card may not follow the actual insert material of the coprocessor.

NOTE: the extra card image that the coprocessor may have requested must be included with the material to be inserted into the assembly. The extra card image, if present, will be the card image examined for the illegal continuation card situation. Any illegal continuation card is made into a comment and an error message is given.

PERTINENT OPERATING FACTORS

The following operating factors influence the previous specifications:

1. After Pass 1 has entered the coprocessor through any of the possible seven entry points, the coprocessor must save the status of the index registers. On re-entry into Pass 1, the coprocessor must restore the previous status of the index registers, excepting index register 7, which now contains new communication information.

2. Once the coprocessor has accepted a condition on an "error" card image, all the remaining information on this "error" card image is lost as far as Pass 1 is concerned except if the coprocessor includes the remaining information of the "error" card on a new card image in a group to be inserted.

3. The END instruction must be the only instruction in its card block whenever STRAP II is running with the indicator, Hhcom4, on.

4. The card image given to the coprocessor under the "WANT ANOTHER CARD" option is in the original input code, not in the A8 processing code of Pass 1.

5. Pass 1 will not have scanned a card image for the END instruction before going to the coprocessor if the card image is one for "WANT ANOTHER CARD" option or one on which an illegal character is present.

6. Coprocessor must scan for an END instruction each card image received under the "WANT ANOTHER CARD" option or under the illegal card input condition. If the coprocessor detects the END pseudo-op, the coprocessor must not ask for any more card images and the subsequent return to Pass 1 must be by the "INSERT CARDS" return. The insert group must at least include the END instruction.

7. Control is given to the coprocessor for conditions present in the original input, not those in an insert group.

For STRAP II to produce relocatable binary output, following pseudo-ops had to be defined: ORIGIN, PUNREL, PUNFPC, ENTER, PUNCDC, COMBLOCK, SLCRCOM, and FEND. The implementation of these pseudo-ops involved introducing both new indicators and new fields to existing formats. The changes to three of the formats, the unit, the symbol table entry, and VALUE's output words are indicated here. Also in this section are both a resume of the processing of the ops, together with the formats of the binary output produced, and the specifications of the pseudo-ops with an example.

## RELOCATION PSEUDO-OPS

| Relocation Pseudo-Op | Specification | Use | Notes |
|---|---|---|---|
| ORIGIN | ORIGIN, N | To produce special origin card for the loader | Where name is not meaningful.<br>Where N=location, in absolute or symbolic form. |
| PUNREL | PUNREL | To put STRAP II in relocatable mode | Where name and address are not meaningful. |
| PUNFPC | PUNFPC, S, C | To produce special Fortran program for the loader | Where name is not meaningful.<br>Where S=program size, in absolute or symbolic form.<br>Where C=blank common size, in absolute or symbolic form. |
| ENTER | A \| ENTER, B | To provide information for the Fortran program card | Where A=any eight characters excluding the quote and semicolon characters.<br>Where B=program entry point, in absolute or symbolic form. |
| PUNCDC | PUNCDC | To produce special common definition card for the loader | Where name and address are not meaningful. |
| COMBLOCK | A \| COMBLOCK, N | To define a named common | Where A=any name (8 characters maximum).<br>Where N=block size, in absolute or symbolic form. |
| SLCRCOM | SLCRCOM, B | To set location relative to common | Where name is not meaningful.<br>Where B if in absolute or symbolic form = named common.<br>Where B if blank = blank common. |
| FEND | FEND | To produce Fortran branch card | Where name and address are not meaningful. |

AN EXAMPLE OF A STRAP II PROGRAM USING RELOCATION PSEUDO-OPS

```
                PUNFPC, LAST, COMLAST
KRUM            ENTER, START
                PUNCDC
BLOCK1          COMBLOCK, 100
                PUNREL
                SLC, 0.0
                DD(BU), 1.0                     'Length of T. V.
INPUT           (A*)DD(BU, 64, 8), INPUT bbb*   'T.V.
XSAV            DR(BU), (15)                    'Index saving calls.
START           TI, 15, $X1, XSAV               'First instruction.
                SIC, $X15
                B, INPUT                        'CALL INPUT
                , A; , A
                , B; , B
                LX, $X14, XW1                   'I index.

                SIC, $X15; B, $MCP              'RETURN.
                , $EOJ
A               DD(N), (10)
LAST            SYN, $
                SLCRCOM
B               DD(N), (10)
COMLAST         DR(N), 0
                SLCRCOM, BLOCK1
C               DD(N), (100)
                FEND
```

---

SOME CHANGES IN THE FORMATS

An additional word had to be added to the basic format of the unit to contain the relocatable information: whether the contents of the address field or of the refill field or of both are to be relocatable, and if so, to what base, See Figure 17.

| Position in Word | Use |
|---|---|
| 0-17 | Number of named common for the address from an address field. |
| 18-23 | Zero fill to force a full word address |

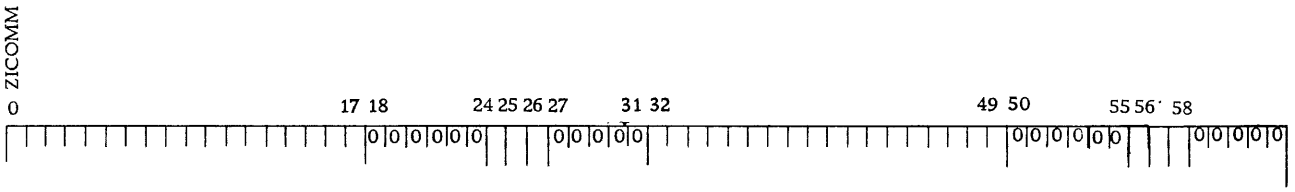



Figure 17. Format of the Last Word in the Fixed Position of a Unit

| Position in Word | Use |
|---|---|
| 24 | 0-if address is not to be relocated.<br>1-if address is to be relocated. |
| 25 | 0-if address to be relocated is from an address field.<br>1-if the address to be relocated is from a refill field. |
| 26 | 0-if address is to be relocated as a lower address.<br>1-if address is to be relocated as an upper address. |
| 27-31 | Not used |
| 32-49 | Number of named common for the address either from a second address field or from a refill field of a full word instruction. |
| 50-55 | Zero fill to force a full word address. |
| 56 | As above. |
| 57 | As above. |
| 58 | As above. |
| 59-63 | Not used. |

Each entry in the symbol table entry also includes fields to flag relocatable information:

| Relative Position in the vle | Use |
|---|---|
| Zst35 | 0-if location counter dependence is not decided.<br>1-if location counter dependence is decided. |
| Zst36 | 0-if location counter dependent.<br>1-if not location counter dependent. |
| Zstcom | 0-if address is to be relocated as a lower address. |
| | 1-if address is to be relocated as an upper address relative to the contents of Zstcon. |
| Zstcon | 0-if blank common otherwise-number of named common. |

When both Zstl3 is on and Zst36 is off, it is necessary to look at Zstcom. If Zstcom is zero, the address is to be relocated as a lower address; if on, the address is to be relocated as an upper address relative to the common whose number is given in Zstcon. When data follows an SLCRCOM statement, Pass 1 sets up each symbol table entry both with Zstcom = 1 and with the common number in Zstcon. Later in Pass 2 after INSERT has inserted the binary output in Ziotpt, INSERT also checks in VALUE's output words the relocation indicators: Sinrl1, Sinrl2, and Sinua. If Sinrl1 = 1, Sinrl2 = 0, and Sinua = 1, then INSERT inserts in the unit at Zicomm the number of the named common contained in VALUE's Sincom.

Figure 18 shows the third word in VALUE's output which contains the extra relocatable information for the user routine.

| Position | Use |
|---|---|
| 0-17 | Number of named common. |
| 18 | 0-if at exit from VALUE, the address is to be relocated as a lower address.<br>1-if at exit from VALUE, the address is to be relocated as an upper address. |
| 19-63 | Not used. |

The fifth word in VALUE's heading is reserved specifically for relocation information during VALUE's evaluating procedure. See Figure 19.

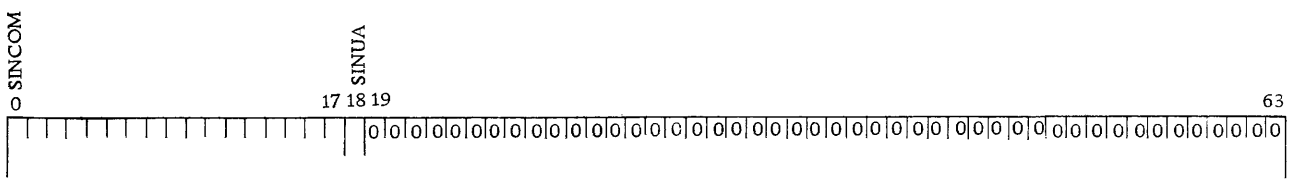



**Figure 18. Format of the Third Word in VALUE's Output**

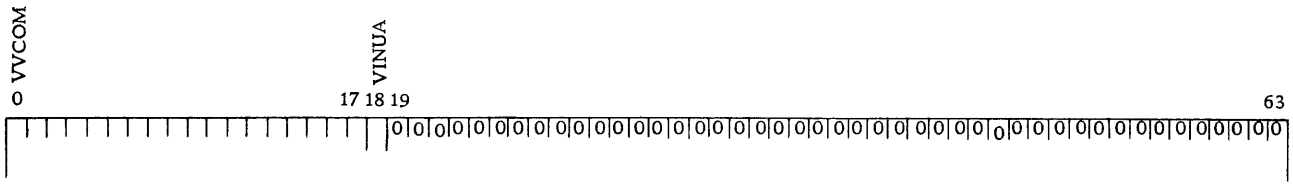



Figure 19. Format of the Fifth Word in VALUE's Heading

The bit assignment is the same as that in the third word in VALUE's output.

RESUMÉ OF THE PROCESSING OF THE RELOCATION PSEUDO-OPS

PUNREL: On encountering a PUNREL pseudo-op, Pass 1 turns on bit Punrel. Subsequent non-PUNREL punch pseudo-ops effectively turn it off so Pass 2 later turns it on also. Pass 2 does not accept relocation pseudo-ops if it is not in PUNREL mode.

PUNFPC: For a PUNFPC, Pass 1 encodes the program and common size fields. If a blank common is not used, the second field may be left blank. If in PUNREL mode, OUTPUT goes into PUNFPC mode and sets up the format for the Fortran Program Card. OUTPUT leaves PUNFPC mode when a non-ENTER statement is encountered.

ENTER: For an ENTER, Pass 1 makes a symbol table entry for the name, if it has one, and encodes the address field. It is OUTPUT's job to determine if it is in PUNFPC mode, to fetch the name (sixty-four bits, A8 code ) from the name file, and to place it in the Fortran Program Card format. If the ENTER does not have a name, OUTPUT will supply eight A8 blanks. OUTPUT also evaluates the entry point address, using VALUE, and places the value in the Fortran Program Card as a nineteen bit field. When either a non-ENTER statement is encountered or the card is full, the Fortran Program Card is punched. Additional Fortran Cards are in the same format except that the program size and amount of blank common used fields are blank.

PUNCDC: If in PUNREL mode OUTPUT goes into PUNCDC mode and sets up a format for the Common Definition Card . OUTPUT leaves PUNCDC mode when a non-COMBLOCK statement is encountered.

COMBLOCK: The COMBLOCK statement does not reserve any data: it simply supplies information for the common block definition card. Data is entered into a named common in the following manner:

Program Example

```
BL1     COMBLOCK, 5
BL2     COMBLOCK, 25
              .          'Program instructions
              .
              .
        SLCRCOM, BL1
JOE     DD(N), 5, 10, 15, 20, 25
        SLCRCOM, BL2
JACK    DRZ(N), $PI
```

References to JOE in the program are marked as relocatable as an upper address relative to common #1.

References to JACK are marked as relative to common #2.

Symbol table entries for COMBLOCK have Zstcom set to one and Zstcon set up to contain the common block reference number as an eighteen bit field. If the statement does not have a name, an error message is given, and the address field is still encoded. This is not the way to define blank common. It should be done thusly:

Program Example

```
        SLCRCOM
JIM     DR(BU), 32
```

References to JIM in the program are each marked as an upper address relative to common #0 (blank common).

It is OUTPUT's job to determine if it is in PUNCDC mode, to fetch the 64-bit A8 name from the name file, to place it in the Common Definition Card format, to evaluate, using VALUE, the common block size, and to place this in the Common Definition Card as an eighteen bit field. When either a non-COMBLOCK statement is encountered or the card is full, the Common Definition Card is punched. Additional Common Definition Cards are in the same format.

SLCRCOM: For a SLCRCOM, Pass 1 turns on bit Mupper to indicate that the symbol table entries made are relative to some upper address. If the address field is blank, the symbol table entries are marked as relative to common #0 (blank). If the address is not blank but contains a legitimate name, it is looked up and the symbol table entry is marked as relative to the common number which is found in the symbol table entry for the name in the address field. Pass 2's job is to set the location counter to zero. In blank common instructions they should be only DR's (no DRZ's).

## FORMATS OF THE RELOCATABLE BINARY OUTPUT PRODUCED BY OUTPUT

STRAP II in PUNREL mode produces binary output as herein described.

### Relocatable Data Card

The Relocatable Data Card contains both data from DD statements and a count of bits to be zeroed or skipped at the time before or after loading from DR or DRZ statements.

| Bits Assigned | Use |
|---|---|
| 1.0 - 1.11 | Code column. (Relocatable data card has 6, 7, 8, and 9 punches.) |
| 2.0 - 2.11 | Identification column. |
| 3.0 - 3.11 | Sequence number. |
| 4.0 - 4.11 | Check sum. |
| 5.0 | A one bit control: 0 if skipping, 1 if setting to zero. |
| 5.1 | A one bit control: 0 if either skipping or zeroing is done before card contents are loaded, a 1 if after. |
| 5.2 - 5.11 | Bit count (10 bits). |
| 6.0 - 7.11 | Origin. |
| 8.0 - 9.11 | Number of bits to be either skipped or zeroed at the time either before or after loading (24 bits). |
| 10.0 - 10.11 | Loading base. |
| 11.0 - 71.11 | Data only. |
| 72.0 - 80.11 | A nine column field ignored by the loader, which may be used for card code identification and sequencing. |

### Common Definition Card

The Common Definition Card contains information regarding named commons for the BSS Loader. The common name and size is derived from the COMBLOCK statements.

| Bits Assigned | Use |
|---|---|
| 1.0 - 1.11 | Code column. (Common definition card has 5, 7, 8, and 9 punches.) |
| 2.0 - 2.11 | Identification column. |
| 3.0 - 3.11 | Sequence number. |
| 4.0 - 4.11 | Check sum. |
| 5.0 - 8.11 | Not used (48 bits). |
| 9.0 - 14.3 | First common name from first COMBLOCK definition card (64 bits). |
| 14.4 - 15.9 | Common size (18 bits). |
| 15.10 - 15.11 | Not used (2 bits). |
| 16.0 - 21.3 | Second common name from second COMBLOCK definition card (64 bits). |
| 21.4 - 22.9 | Common size (18 bits). |
| 22.10 - 22.11 | Not used (2 bits). |
| 23.0 - 29.11 | The above pattern is repeated for third common name. |
| 30.0 - 36.11 | The above pattern is repeated for fourth common name. |
| 37.0 - 43.11 | The above pattern is repeated for fifth common name. |
| 44.0 - 50.11 | The above pattern is repeated for sixth common name. |
| 51.0 - 57.11 | The above pattern is repeated for seventh common name. |
| 58.0 - 64.11 | The above pattern is repeated for eighth common name. |
| 65.0 - 71.11 | The above pattern is repeated for ninth common name. |
| 72.0 - 80.11 | A nine column field ignored by the loader, which may be used for card code identification and sequencing. |

### Fortran Branch Card

The Fortran Branch Card is produced as a result of the FEND pseudo-op.

| Bits Assigned | Use |
|---|---|
| 1.0 - 1.11 | Code column. (Fortran branch card has 5, 6, 7, 8, and 9 punches.) |
| 2.0 - 2.11 | Identification column. |
| 3.0 - 3.11 | Sequence number. |
| 4.0 - 4.11 | Check sum. |
| 5.0 - 71.11 | Not used. |
| 72.0 - 80.11 | A nine column field ignored by the loader, which may be used for card code identification and sequencing. |

Fortran Program Card

The Fortran Program Card contains information for the BSS Loader regarding both the size of blank common and entry points derived from ENTER statements.

| Bits Assigned | Use |
|---|---|
| 1.0 - 1.11 | Code column. (Fortran program card has 5, 6, 7, and 9 punches.) |
| 2.0 - 2.11 | Identification. |
| 3.0 - 3.11 | Sequence number. |
| 4.0 - 4.11 | Check sum. |
| 5.0 - 5.11 | Not used. |
| 6.0 - 7.5 | Program size (18 bits). |
| 7.6 - 8.11 | Size of blank common (18 bits). |
| 9.0 - 14.3 | A8 name from first ENTER statement (64 bits). |
| 14.4 - 15.10 | Entry address (19 bits). |
| 15.11 | Not used (1 bit). |
| 16.0 - 21.3 | A8 name from first ENTER statement (64 bits). |
| 21.4 - 22.10 | Entry address (19 bits). |
| 22.11 | Not used (1 bit). |
| 23.0 - 29.11 | The above pattern is repeated for the third entry point name. |
| 30.0 - 36.11 | The above pattern is repeated for the fourth entry point name. |
| 37.0 - 43.11 | The above pattern is repeated for the fifth entry point name. |
| 44.0 - 50.11 | The above pattern is repeated for the sixth entry point name. |
| 51.0 - 57.11 | The above pattern is repeated for the seventh entry point name. |
| 58.0 - 64.11 | The above pattern is repeated for the eighth entry point name. |
| 65.0 - 71.11 | The above pattern is repeated for the ninth entry point name. |
| 72.0 - 80.11 | A nine column field ignored by the loader, which may be used for card code identification and sequencing. |

Relocatable Binary Instruction Card

The Relocatable Binary Instruction Card contains instructions, XW's, CF's, and RF's only (i.e., no DD's or DR's) and their respective relocation bits. The bit count in column five contains the count of bits to be loaded, excluding the relocation bits.

| Bits Assigned | Use |
|---|---|
| 1.0 - 1.11 | Code column. (Relocatable binary instruction card has 5, 7, and 9 punches.) |
| 2.0 - 2.11 | Identification column. |
| 3.0 - 3.11 | Sequence number. |
| 4.0 - 4.11 | Check sum. |
| 5.0 - 5.11 | Bit count. |
| 6.0 - 7.11 | Origin |
| 8.0 - 8.11 | Not used. |
| 9.0 - 71.11 | Variable binary output for instructions only, followed by variable relocation bits. |
| 72.0 - 80.11 | A nine column field ignored by the loader, which may be used for card code identification and sequencing. |

Relocation Bits

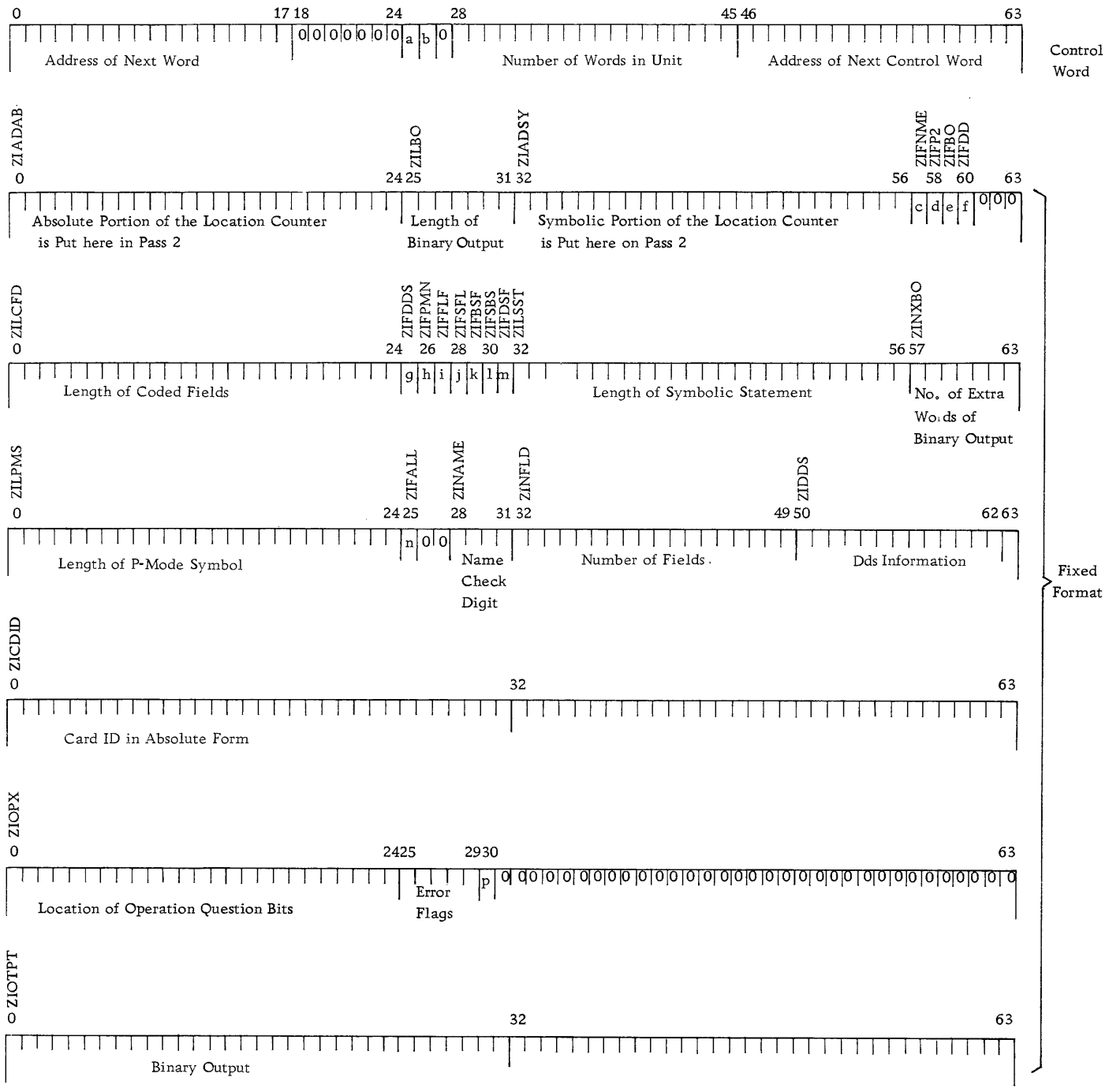
| | |
|---|---|
| 0 | No relocation |
| 1 0 .. .. | Relocation |
| 1 0 0 .. | First 18 bits (address) |
| 1 0 1 .. | Last 18 bits (refill) |
| 1 0 .. 0 | As lower address |
| 1 0 .. 1 .. | As upper address |
| 1 0 .. 1 0 | Blank common |
| 1 0 .. 1 1 i | Named common where i is the number of the named common. Length of i field is determined by the number of named commons. |

This bit is reserved for later definition.

Special Origin Card

The Special Origin Card for the BSS Loader has the following format:

| Bits Assigned | Use |
|---|---|
| 1.0 - 1.11 | Code column. (Special origin card has an O- punch.) |
| 2.0 - 9.11 | New origin. (The full word location is punched in the format: xxxxxx.$0_8$.) |
| 10.0 - 71.11 | Not used. |
| 72.0 - 80.11 | A nine column field ignored by the loader, which may be used for card code identification and sequencing. |




Legend

a. on if last control word
b. on if last control word in file
c. name
d. DECODE gets unit
e. binary output
f. DD
g. explicit dds
h. P-mode
i. f.1. filled in
j. symbolic f.1.
k. b.s. filled in
l. symbolic b.s.
m. dds filled in
n. nothing left for DECODE
p. suppress e.m.

*See section on relocation pseudo ops.

Figure 20. Format of the Fixed Portion of an Expanded Instruction Unit (not including relocation fields)*

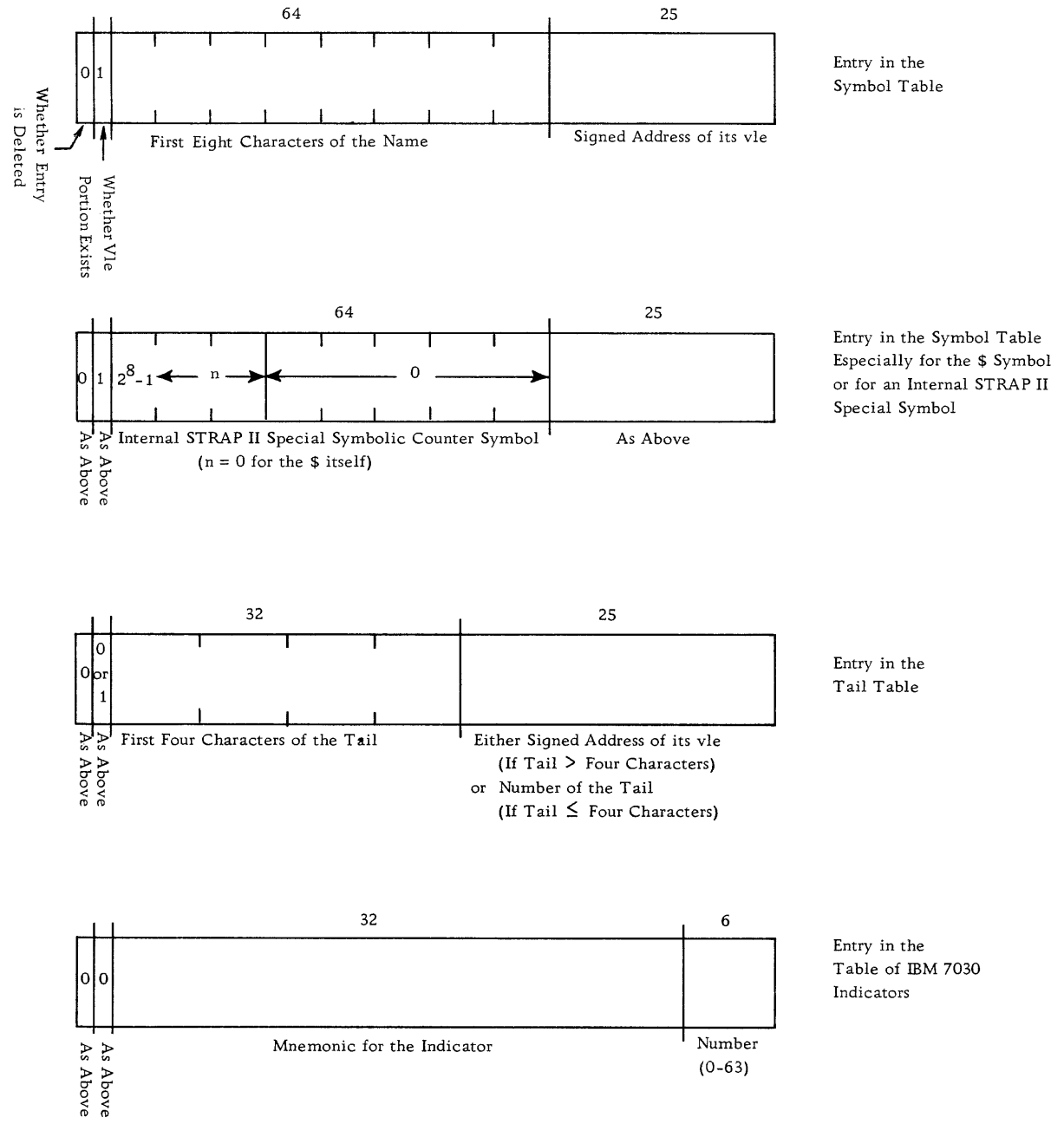



Figure 21. Augment Portion of Each of the Tables Handled by Table Management

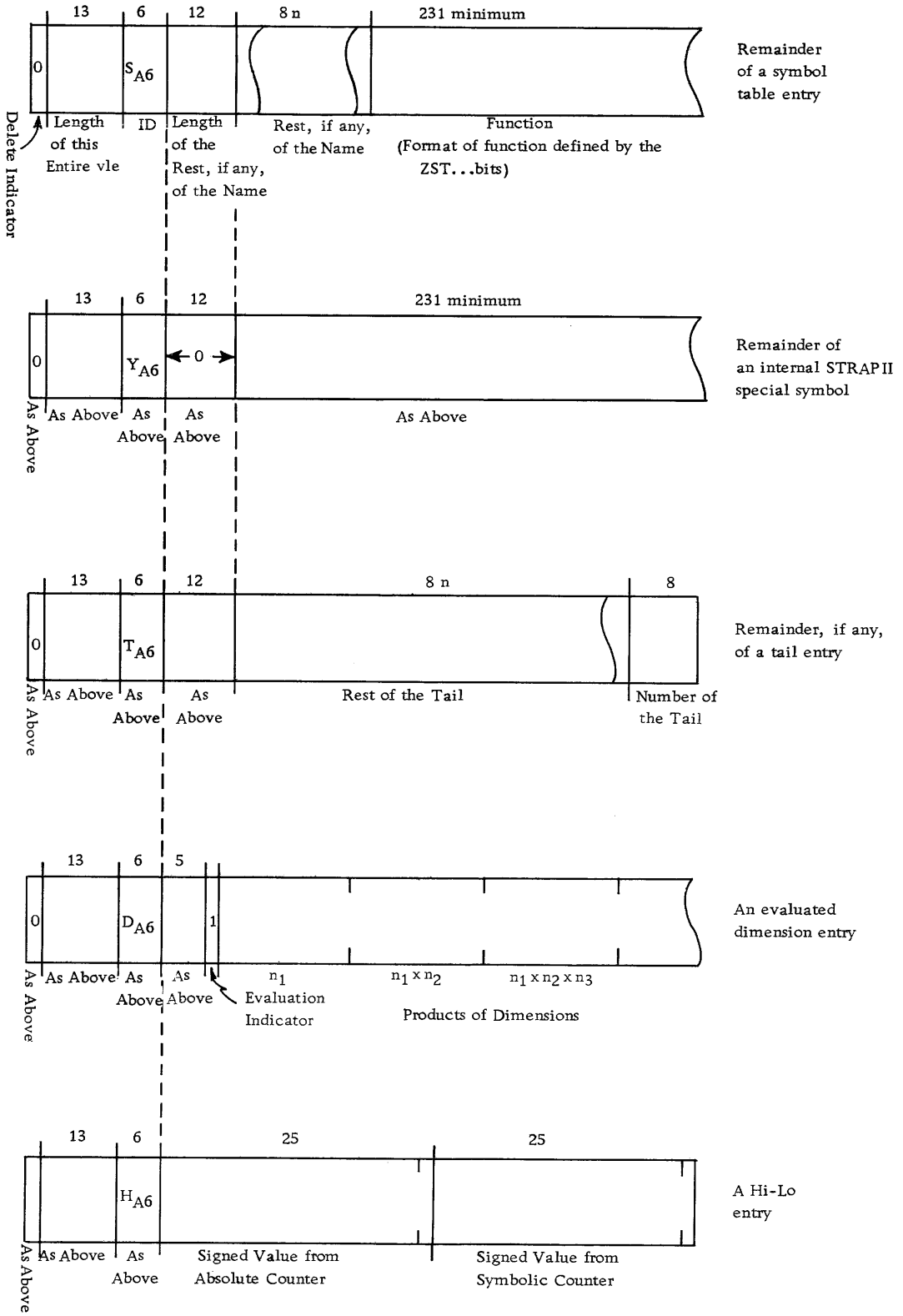
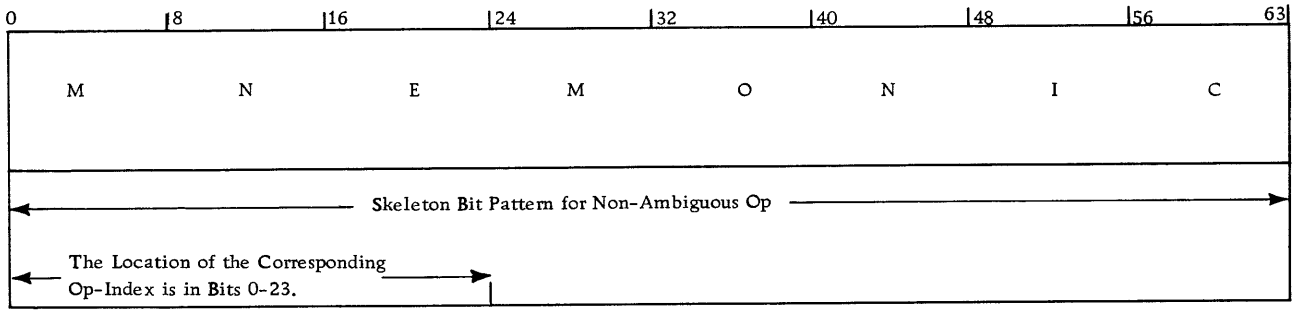
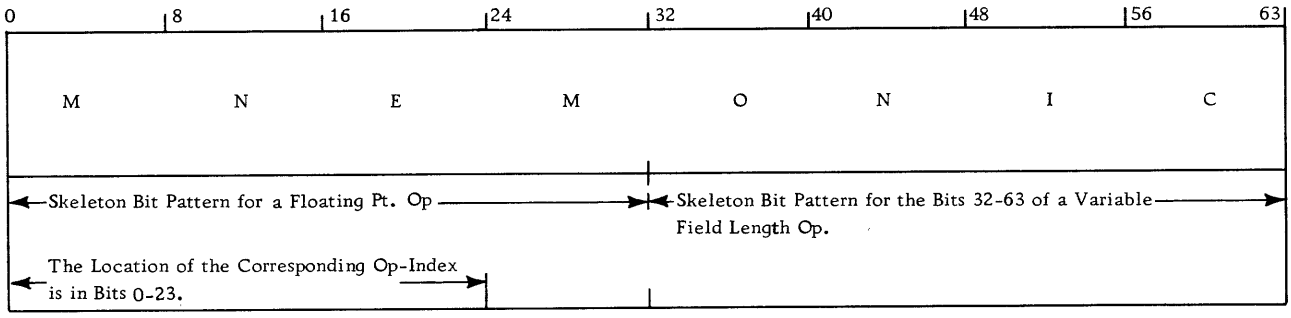


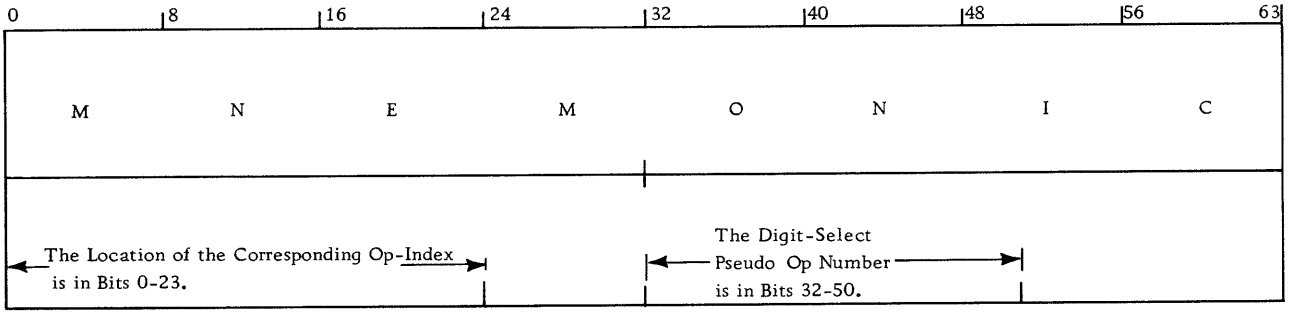

Figure 22. Entries in the Variable Length Entry (vle) Table




Two Word Format for a Non-Ambiguous Op

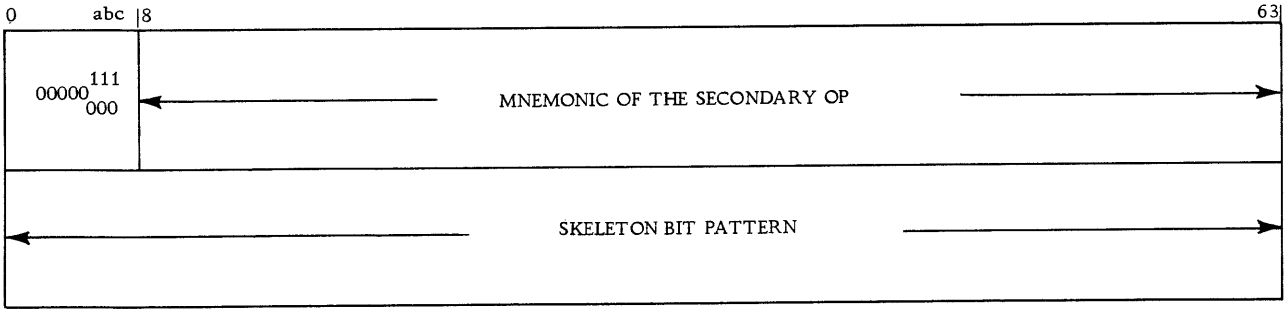



Two Word Format for an Ambiguous Op



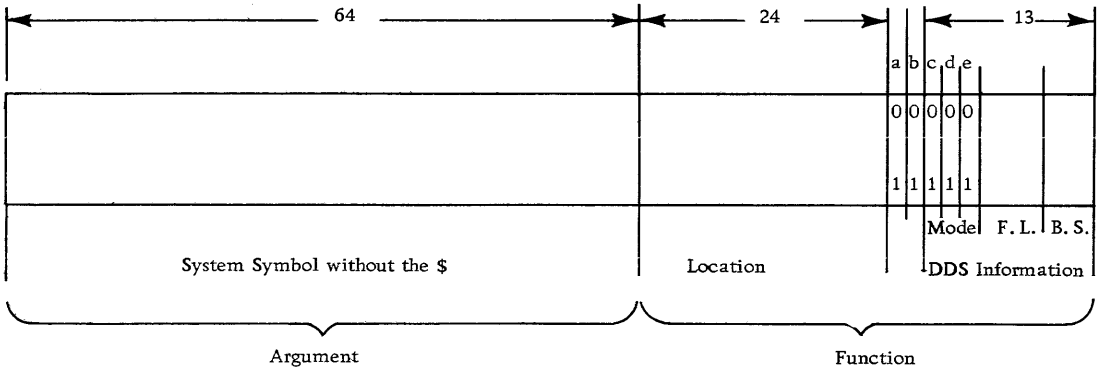

Two Word Format for a Pseudo Op

Figure 23. Entries in the Table of Primary Operations




| abc | Use |
|---|---|
| 000 | - |
| 001 | for a secondary op in a variable field length instruction |
| 020 | for a SEOP |
| 100 | for a secondary op in the CW instruction |

Figure 24. Entry in the Table of Secondary Operations




a = 0, if the location is the machine location corresponding to the system symbol.
a = 1, if the location is an address within STRAP II where the floating point constant for the system symbol is stored.

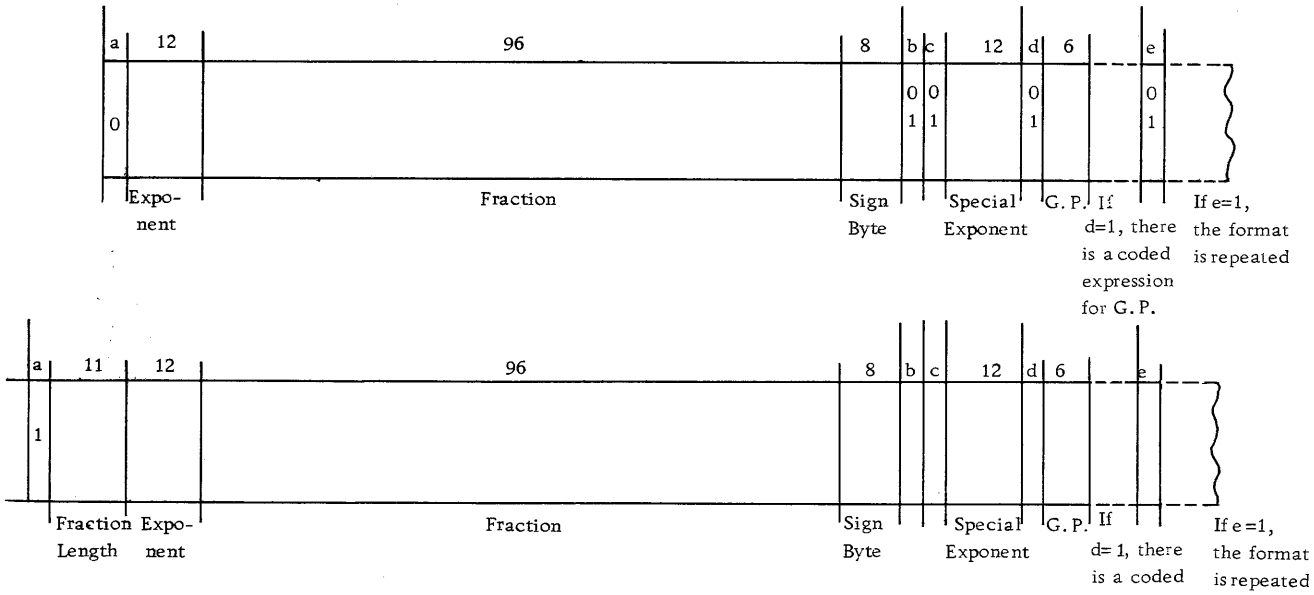
b = 0, if the location for the system symbol is available at this IBM 7030 machine installation.
b = 1, if the location for the system symbol is not available at this IBM 7030 machine installation.

c = 0, if mode is floating point.
c = 1, if mode is not floating point.

d = 0, if mode is either unnormalized or unsigned.
d = 1, if mode is either normalized or signed.

e = 0, if mode is decimal.
e = 1, if mode is binary.

Figure 25. Entry in the Table of System Symbols




a = 1, if the numeric DD is in the Fn entry mode.
a = 0, if the numeric DD is not in the Fn entry mode.

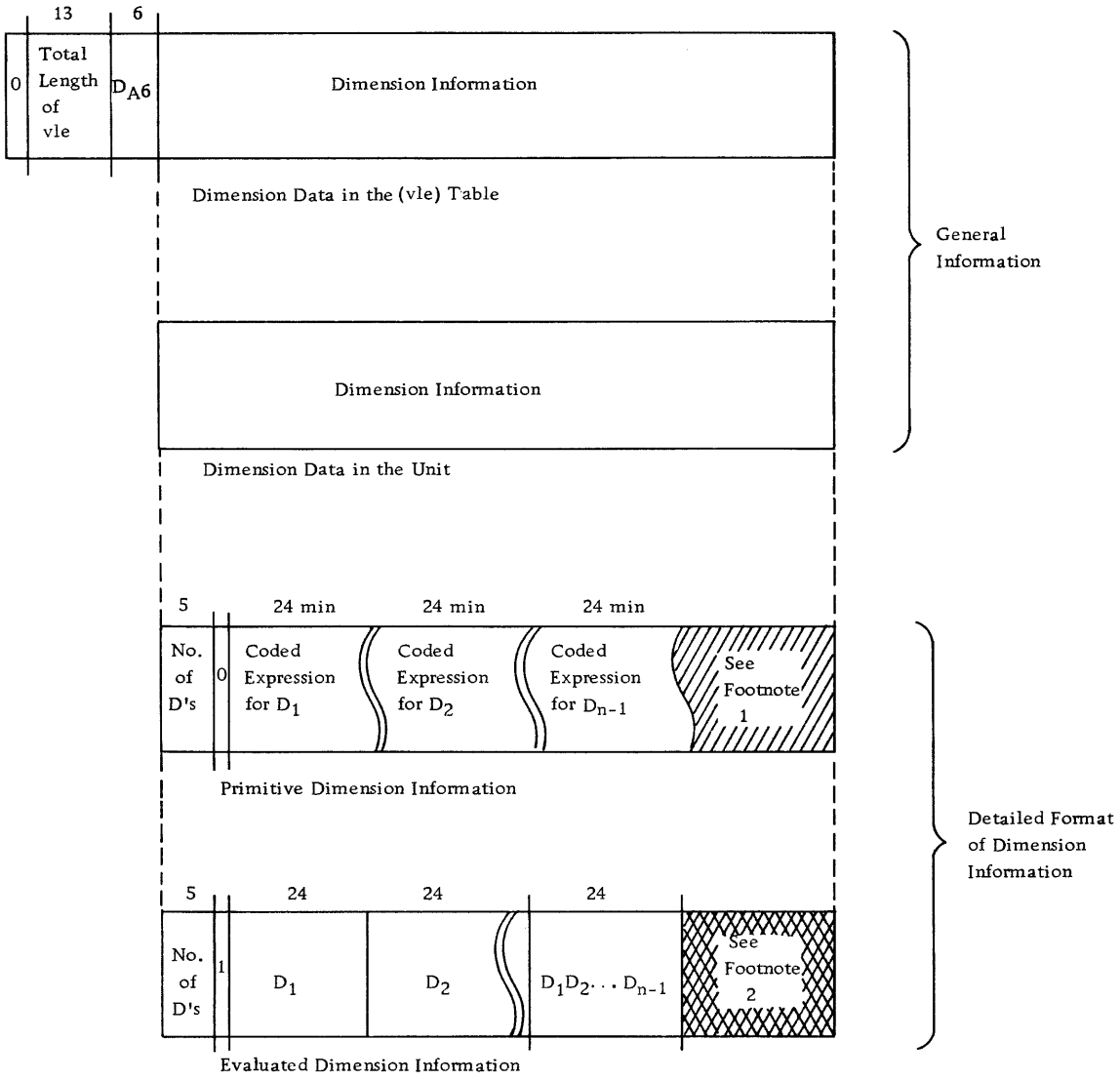
b = 1, if the numeric DD is in the normalized mode.
b = 0, if the numeric DD is not in the normalized mode.

c = 1, if the numeric DD has a special exponent.
c = 0, if the numeric DD does not have a special exponent.

d = 1, if the numeric DD has a general parenthetical field integer entry (G.P.).
d = 0, if the numeric DD does not have a general parenthetical field integer entry.

e = 1, if the format is repeated for another field in a multiple field numeric DD.
e = 0, if this is the end of the encoding.

Figure 26. Entry Made in the Variable Portion of a Numeric DD's Expanded Instruction Unit




1. This is 'garbage' in case it is necessary to make the total length of the primitive form $\geq 24(n-1)+6$ bits to allow for the conversion of the primitive to the completed form.
2. This is 'garbage' if the primitive form was longer than $24(n-1)+6$ bits. (The transition from the primitive to the completed form is handled by the VALUE subroutine when the dimension reference is required.)

Figure 27. Dimension Entry

## APPENDIX B--ADDITIONAL INSTRUCTIONS AND PSEUDO-OPS ACCEPTED BY STRAP II[1,2]

PRIMARY OPERATIONS

### Pseudo-Ops[3]

| | | | |
|---|---|---|---|
| COMBLOCK | Common Block Definition | PUNFPC | Punch FORTRAN Program Card |
| CNOP | Conditional No Operation[4] | PUNFUL | Punch Full Binary Card(s) |
| DDI | Data Definition Immediate | PUNID | Punch ID in Binary Card(s) |
| DR | Data Reservation | PUNNOR | Punch Normally |
| DRZ | Data Reservation and Set to Zero | PUNORG | Punch Origin Binary Card(s) |
| DUPLI | Duplicate | PUNREL | Punch Relocatable Binary |
| END | End | PUNSYN(or PUNSYM) | Punch Syn Symbolic Card(s) |
| ENTER | Define Entry Point | REM | Restore Error Message |
| EXT | Extract[4] | RESEQ | Restore Punching Sequence Number in Binary Card(s) with one |
| FEND | FORTRAN End | SEM | Suppress Error Message |
| LINK | Link | SKIP | Skip Paper |
| NOPRINT | No Printing of Listing | SLC | Set Location Counter |
| NOPUN | No Punching of Binary Output | SLCR | Set Location Counter Relative |
| NOSEQ | No Sequence Number to be Punched in Binary Card(s) | SLCRCOM | Set Location Counter Relative to Common |
| ORIGIN | Origin | SPNUS | Suppress Printing Not Used Symbol List |
| PRND | Print double-Spaced | SYN | Synonym |
| PRNID | Print ID | TAIL | Tail |
| PRNOR | Print Normally | TLB | Terminate Loading and Branch |
| PRNS | Print Single-spaced | UNTAIL | Untail |
| PRNTALL | Print All Symbol(s) Used in Program | | |
| PUNALL | Punch SYN Card(s) for All Symbol(s) | | |
| PUNCDC | Punch Common Definition Card | | |

### General Instructions

| | |
|---|---|
| CF | Count Field |
| CW | Control Word |
| DD | Data Definition |
| INDMK | Indicator Mask |
| RF | Refill Field |
| VF | Value Field |
| XW | Index Word |

### MCP Instructions

IOD
REEL

Input-Output Instructions: OP, $A_7(I)$ where $A_7(I)$ represents a channel address and the unit affected is the last unit selected by an LOC instruction.

| | |
|---|---|
| BS | Backspace |
| BSSEOP | Backspace, Suppress End of Operation Interrupt |
| BSFL | Backspace File |
| BSFLSEOP | |
| ECC | ECC (and odd parity for tape) |
| ECCSEOP | |
| ERG | Erase Gap |
| ERGSEOP | |
| EVEN | Even Parity, No ECC (tape only) |
| EVENSEOP | |
| GONG | Sound Gong |
| GONGSEOP | |
| HD | High Density |
| HDSEOP | |
| KLN | Check Light On |
| LD | Low Density |
| LDSEOP | |
| NOECC | No ECC, EVEN Parity (tape only) |
| ODD | Odd Parity, No ECC |
| ODDSEOP | |
| ODDECC | Odd Parity, ECC |
| ODDNEC | Odd Parity, No ECC |
| RLF | Reserved Light Off |
| RLFSEOP | |
| RLN | Reserved Light On |
| RLNSEOP | |
| RWDUNL | Rewind and Unload |
| SP | Space |
| SPSEOP | |
| SPFL | Space File |
| SPFLSEOP | |
| TLB | Terminate Loading and Branch |
| TILF | Tape Indicator Light Off |
| UNLOAD | Unload |

SECONDARY OPERATIONS

| | |
|---|---|
| CCR | Chain Counts within Record |
| CD | Count Disregarding Record |
| CDSC | Count Disregarding Record, Skip, and Chain |
| CR | Count within Record |
| SCCR | Skip, Chain Counts within Record |
| SCR | Skip |
| SCD | Skip, Count Disregarding Record |
| SCDSC | Skip, Count Disregarding Record, Skip, and Chain |

1 Additional is used in the sense that the instructions and pseudo-ops are not specified in the IBM-7030 Reference Manual.
2 An operation mnemonic cannot exceed 8 characters in length.
3 These pseudo-ops are in each of STRAP II's three-digit select pseudo-op tables.
4 With the exception of the EXT and CNOP instructions, STRAP II considers an instruction to be a pseudo-op if it does not produce binary output.

## PARTICULAR COMPOSITION OF A CODED EXPRESSION

Any coded expression begins with a 4 bit prefix, of which the first bit is 1 (to indicate the existence of the coded expression). The format of the coded expression depends on the prefix.

1000 -- Normal Case. Following this prefix there is a list of data, terminated by a 0-bit (after the 0-bit terminating the last datum); then there is a list of operators, again terminated by a 0-bit. Finally the entire coded expression is terminated by a 0-bit.

The operators are in the order in which they are to be performed.

1001 -- Special Index Expression. Following this prefix and an additional three bits is the same encoding as that after a normal case prefix. The final result is to be considered as an index (with a basic value of 0).

1010 -- Padded Case. Following this prefix is the same encoding as that after a normal case prefix, but after the last 0 of the normal case type encoding is a padding, that is, an arbitrary number of 1 bits and a final 0 bit to end the expression.

1011 -- Special Signed Value. After the prefix is a 25 bit (sign included) value followed by a padding that is an arbitrary number of 1-bits and a 0-bit.

1100 -- Super Special. (Normal case that has been finished by DECODE.) Following this prefix is the same encoding as that after the normal case prefix. (This super special prefix is intended to indicate to super-programs, e.g., DECODE, that the information in the coded expression has already been used.)

1101 -- Fully Evaluated Expression. Following the prefix are two full words (128 bits) giving the values, etc., in the same format as that of the output from VALUE.

1110 -- Long Prefix. This prefix indicates that three more bits are to be considered as part of the prefix for the coded expression.

1110000 -- Integer Value. Following this prefix is the same encoding as that after a normal case prefix, but if the encoding has a non-zero bit style part, this is to be added into the integer part, and an error message is to be given.

1110001 -- Absolute. Following this prefix is the same encoding as that after a normal case prefix, but after computation of the coded expression, the values are set positive.

1110010 -- Absolute Integer. This is like the integer value and absolute cases in that order.

1110011 -- 17-Bit Constant. Following this prefix is a 17-bit number and padding. The number is used for the I value, and the S/X portion is taken as S. (This prefix is intended for internal use by VALUE in dealing with subscripts. The length '17' is chosen because this type of expression is now short enough to replace any other.)

1111 -- Null Expression. When this prefix appears in subscript position after a datum, this prefix terminates the datum (i.e., it has the same effect as a single 0-bit). When this prefix appears as a field in its own right it represents a null field, and is followed by padding as in padded and special signed value cases (i.e., it terminates at the first 0-bit).

Any encoding of a datum begins with a prefix of three bits, of which the first bit is 1 (to indicate a datum exists). The format of the encoding for the datum depends on this 3-bit prefix.

100 -- Symbolic Datum. This prefix is followed both by seven bits giving the number of A8 bytes (characters) in the symbol and then by the symbol in A8 code.

110 -- Absolute Datum. This prefix is followed by a 24-bit binary number (interpreted as integer part).

101 -- Sub-Expression. This prefix is followed by a coded expression whose values are to be used for the datum.

111 -- Long Prefix. This prefix indicates that two more bits are to be looked at as part of the prefix to the encoding for the datum.

11100 -- System Symbol Datum. This prefix is followed by a 24-bit binary number (interpreted as the bit style part) and by 13 bits of data description information.

11101
11110 -- Not used.
11111

Encoding for a subscript, if present, is that of a regular coded expression. The encoding for a subscript immediately follows the encoding of the symbol to which the subscript refers.

Encoding for an index, if present, is that of a regular coded expression. The encoding for an index follows the encoding of the last datum and precedes the encoding of the operators.

Encoding for operator, if present, in the field consists of nine bits of which the first bit is 1 (to indicate operators exist). The nine bits are made up of the following fields: a 1-bit, a 3-bit operator code, and a 5-bit datum number telling which datum the operator follows. The operator codes are:

| | |
|---|---|
| 111 | .+, a . in a context such as 2.3, A.B. |
| 110 | .L, a . left in a context such as 2., 2. +3. |
| 101 | .R, a . right in a context such as .2, 2+.3. |
| 100 | U-, a unitary - in a context such as -7, A*-2. |
| 011 | * |
| 010 | / |
| 001 | + |
| 000 | -, a - in a context such as 2-3, A-2. |

Encoding for a general parenthetical field entry, if present, includes the 1-bit (to indicate a general parenthetical field entry exists) followed both by six bits giving the value of N and by a coded expression for the entry to be OR'ed into the field. The encoding for a general parenthetical field entry follows encoding of the operators.

GENERAL COMPOSITION OF A CODED EXPRESSION

| Item Number for Referencing Purposes | Bit Length | Comments |
|---|---|---|
| **Prefix Portion (once for every coded expression):** | | |
| 1A | 1 | Indicator: 1-if coded expression exists; 0-if no (more) expression(s). |
| 1B | 3 | Type of coded expression (normally 000).[1] |
| **Datum Portions:** | | |
| 2A | 1 | Indicator: 1-if datum exists; 0-if no (more) datum. |
| 2B | 2 | Indicator: 10-if absolute datum; 00-if symbolic. |
| 2C or | 24 | Absolute datum: value, binary. |
| 2C and | 7 | Symbolic datum: length of item 2D (symbol) in 8-bit bytes. |
| 2D | Variable | Symbol (in A8 code). |
| 2E | 1 | Indicator[2]: 1-if coded expression(s) for symbol subscript exists; 0-if no (more) subscript(s). ("subscript" may be index.) |
| NOTE: Items 2E and 2F are repeated for each subscript until item 2E is "0." | | |
| 2F | Variable | Coded expression for subscript field.[3] |
| NOTE: Items 2A through 2E(F) are repeated for as many data as exist until item 2A is "0." | | |
| **Operator Portions:** | | |
| 3A | 1 | Indicator: 1-if operator exists; 0-if no (more) operator(s). |
| 3B | 3 | Operation code.[4, 1] |
| 3C | 5 | Reference number to datum to which operator applies. |
| NOTE: Items 3A through 3C are repeated for as many operators as exist until item 3A is "0." | | |
| **General Parenthetical Integer Entry Portions (g.p.):** | | |
| 4A | 1 | Indicator: 1-if (.n) entry follows; 0-if no (more) (.n) entries.[5] |
| 4B | 6 | Value of n, binary. |
| 4C | Variable | Coded expression to be OR'ed in. |
| NOTE: Items 4A through 4C are repeated for as many g.p.'s as exist until item 4A is "0." | | |

Examples of a Coded Expression

Figures 28 and 29 illustrate coded expressions.

Footnotes

1. See the detail write-up and example for definitions of these codes.

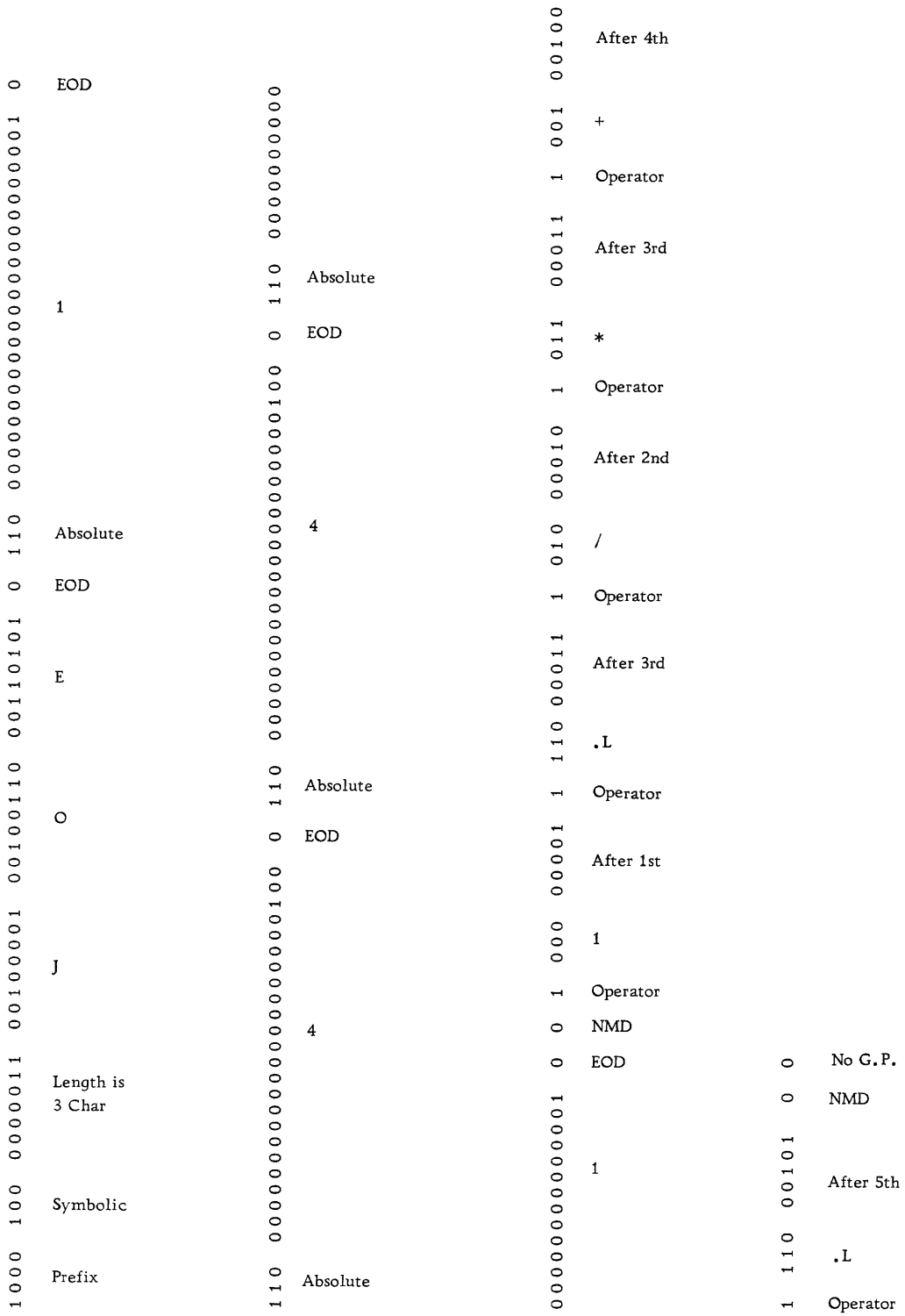
2. This item is actually 1A of a coded expression within the coded expression. Voilà la cereal box! If this seems confusing, think of the box of cereal that has a picture of a boy holding a box of cereal with a picture of a boy holding a box of cereal with a picture .... This "cereal box" idea is a key to both the formats, and the subroutines which create and interpret them.

3. Each datum is followed by a "0", the end of a datum indicator, or by more complete coded expressions representing subscripts. The last datum is followed by a "0", the end-of-datums indicator, in addition to end-of-datum indicator. Since each expression starts with a 1-bit, a "0" end of datums indicator is all that is needed to show that there are no more expressions and that datum(s) is (are) complete.

4. Priority of an operator is defined so that the more parentheses the operator is inside, the higher its priority. Within the same pair of parentheses, the order is essentially that given in the table.

5. Actually, the 0-bit setting of item 4A is equivalent to the 0-bit setting of item 1A in its own coded expression.

(((JOE - 1) /4.) * 4 + 1).


1000 100 0000011 001000001 001001100 001101101 0 110 0000000000000000001 0
Prefix
Symbolic
Length is 3 Char
J
O
E
EOD
Absolute
1
EOD
110 0000000000000000001 0 110 0000000000000000100 0 110 0000000000000000100 0 110 0000000000000000100

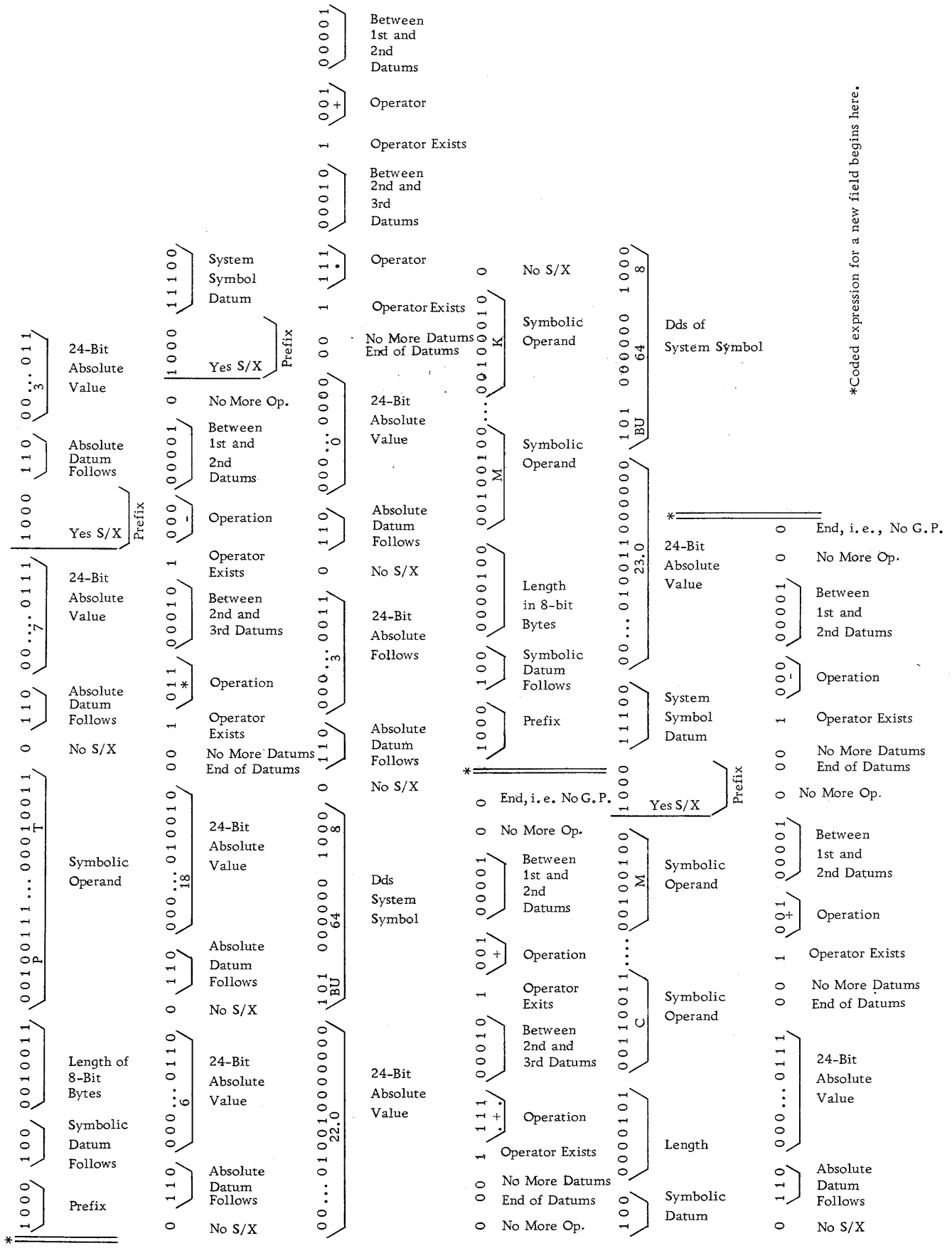



Figure 29, Coded Expression -- Example 2

## APPENDIX D - SYMBOLIC DESIGNATIONS

If any named modifications are to be made to the STRAP II program or if a coprocessor subroutine is to be added to the assembly program, the following mnemonic system already adopted by STRAP II should be noted to avoid possible multiply-defined symbols.

| Initial Character(s) | Used in the Symbolic Locations |
|---|---|
| A | Table management subroutine, e.g., ADDEND, ADDORD |
| AZZ | Triggers |
| AZ | Relative Location in Table Control Block (SYN) |
| AY | Temporary Erasable in Program |
| AX | Bit Lengths in Entries (SYN) |
| AW | Relative Locations in Entries (SYN) |
| AU | Locations in Universal Control Block |
| AS | Locations for Saving Indexes, etc. |
| B | Buffers and index words referring to the buffers. |
| CC | CCA8 |
| CP | CPLTSY, CPLTNM |
| D | Processing of DD fields |
| E | Error message subroutines, e.g., ERR, ERRIN, ERRNUM, and ERRPRT |
| EF | Error flags |
| F | Control flags |
| GC | GETCHA |
| GF | GETFLD |
| H | Interrupt subroutines |
| I | Intermediate input and intermediate output subroutines, e.g., ININIT, INTIN, and INTOUT |
| J | MOVE |
| K | DECODE |
| LL | BOPSER |
| LLL | MBSPEC |
| M | Main flow of Pass 1 |
| N | Main flow of Pass 2 |
| O | OUTPUT |
| P | - - - - |
| QC | Character question (256-bit) strings |
| R | - - - - |
| S | Insertion subroutines |
| T | Tables and setup routines |
| TOPA | Operation fill control words for INSERT |
| TOPQ | Operation question (64-bit) strings |
| U | UITER |
| V | VALUE |
| W | - - - - |
| X | I-O subroutines, e.g., XERR |
| ZI... | SYN's defining the relative locations in an intermediate expanded instruction unit |
| ZO... | SYN's defining the relative locations in the operation question bits |
| ZST... | SYN's defining the relative locations in the function portion of the variable length entry part of a symbol table entry |

Following is a list of terms and abbreviations which are used frequently in the analysis manual, including a number which have certain specialized meanings when used with respect to the STRAP II program. The definitions given here are not exhaustive and the reader is referred to the text of the manual for amplification when required.

Absolute and Symbolic Counters (portion of the location counter)

The location counter setting is maintained through Pass 2 in two portions, each in an index register. $4 contains the absolute setting of the counter; $6 contains the special symbol table entry, if any, which has the special coded expression with the sum of the current symbolic counter and of the absolute counter. Thus, a cascading of STRAP II special symbols can be built-up to refer to the actual setting of the location counter until the absolute length of the binary output is known. (The settings of $4 and $6 are also carried in each unit and in the vle portion of each symbol table entry.)

Ambiguous Op

This is an instruction that may have a floating point dds or a variable field length data description.

Bit Style Number

This is a number which has a decimal point. Since $3 is considered a bit style number and may be written 3.0, a bit style datum may also be called an X-datum.

b. s.

This stands for byte size.

B-value

This stands for bit value.

Card Block

This consists of the group of instructions included on the initial card and on the continuation cards immediately following the initial card.

Coded Expression

This is the encoding form (done primarily by GETFLD) of an expression in a field. See Appendix C.

COMREC

This stands for Communication Record.

Continuation Card

This is a card which has a continuation mark, an asterisk, in the first column.

dds

This stands for data description.

dds reference address

This is a field in the Zst bits that refers to the address of the explicit data description information to be associated with this symbol table entry.

Deck

This is a group of symbolic or binary cards.

Dimension or Subscript

This is an integer specified within parentheses after a symbol to refer to an item within an array.

Dimension Reference Address

This is a field in the Zst bits that refers to the count position of the dimension information usually contained in the same symbol table entry further out in the variable length entry part.

Expanded Instruction Unit

This is a record built-up for each instruction primarily by Pass 1 in which is placed the following information for each statement: control word for INTOUT, information about the instruction, the encoding of the instruction, and the symbolic statement of the instruction.

Expression

A STRAP II expression (constructed according to the rules for the field of the instruction in which the expression is located) is a sequence of constants and of variables, subscripted or not subscripted, separated by operation symbols and parentheses, and followed by either an index or by a general parenthetical field integer entry, but not both.

Fill Control Words

They are a chained list of index words for each instruction. These XW's are used by INSERT's subroutines for inserting the values into the binary output of the instruction.

f. l.

This stands for field length.

General Parenthetical Field Integer Entry

This is the method of OR'ing a pattern of bits into an instruction or into a pseudo instruction which produces binary output. Such pseudo instructions as SYN, DR, DDI, and DRZ cannot have a gp.

g. p. (or gp, or GP)

This stands for general parenthetical integer entry.

Initial Card of a Card Block

This is a card which does not have a continuation mark in the first column.

Integer Style Number

This is a number which does not have a decimal point. Since a subscript is an example of this, an integer style datum may also be called an S-datum.

I-value

This stands for integer value.

Location Counter Dependent Symbol

This is a symbol whose value in the symbol table entry depends on the location counter.

Name

This consists of the legal name characters appearing in columns 2-9 of a card block including the embedded blanks.

Op Index

In STRAP II there is a table of op index words. For each type of instruction, there is a corresponding op index or question bit string. The op index contains:

1. Location of fill index word chains for this type of instruction.
2. Masked set of bits whose status indicate answers to certain yes-no questions about this type of op.
3. A code number if the op is a secondary op.

Pass 1

This is the phase of STRAP II that processes the input statements and builds up an intermediate expanded instruction unit for each instruction, a symbol table, and a name file.

Pass 2a

This is the phase of STRAP II that assigns memory addresses to location counter dependent statements.

Pass 2b

This is the phase of STRAP II that checks memory assignments, completes binary output, and produces final documents.

Phoney Symbol Table Entry

If a name is attached to an operation which does not produce binary output and which is not an MCP op, a SYN, DR, nor a DDI op, a "phoney" or specially marked entry is made in the STRAP II symbol table. Then the instruction will receive an error message for the unnecessary name. Note that address reference to this symbol will receive the setting of the location counter of the next instruction (with or without possible truncation).

Primitive Dimension Reference Entry

This is the initial unevaluated entry made for a dimension specified on a DR or DRZ either in the unit or in the vle table. See Appendix A for format.

Priority of Operations

When arithmetic operations are specified in an instruction field, there is a definite order by which they are combined. The following describes the operations in descending order of combination.

1st order operation + and -.
2nd order operation * and /.
3rd order operation **.

Print Buffer

Listing data that is to be printed by OUTPUT is built up in the print buffer.

Pseudo-Defined Symbol

If a symbol is undefined in the program, the symbol will be assigned a data description of (N), and a location of the next available full word above the upper limit of the assembled program. Such a symbol will be frequently referred to as a pseudo-defined symbol.

Pseudo-Op

STRAP II considers an instruction a pseudo-op if binary output is not to be inserted in STRAP II's expanded instruction unit for the instruction. Generally, this means that if the instruction does not produce binary output, STRAP II considers the instruction a pseudo-op.

Punch Buffer

When enough binary output for a binary card has been accumulated in the intermediate binary storage buffer, OUTPUT transfers the binary output to the punch buffer.

S-Datum

See integer style number.

Serious Error Messages

Those are the numbered error messages 1-28 which indicate the most serious error conditions detected in the program being assembled.

Special Floating Point Ops

These are the floating point operations, E and E+I, to which is assigned a data description of U by DECODE if the op does not have an implicit or explicit dds stated.

STRAP II Special Symbol

STRAP II makes one of its 24-bit special symbols in the symbol table whenever an internal cross referencing symbol is needed. The first eight bits of the STRAP II special symbol is always eight one bits; the remaining sixteen bits are determined at the time the symbol is made. The STRAP II special symbol -- 111111110000000000000000 -- is the one made for a programmer's $ symbol. STRAP II also makes these symbols during the establishing of the location counter value in Pass 2.

Symbol

This consists of the legal name characters appearing in columns 2-9 of a card block excluding the embedded blanks.

TOE

This stands for table of exits.

Types of dds's

The types of dds's will be defined by example.

Explicit dds

Regardless of the dds associated with the symbol, CONSTANT, in the instruction --
L(BU, 64), CONSTANT ---,
the explicit data description for the Load instruction is clearly, (BU, 64).

Implicit dds

When CONSTANT has been defined as --
CONSTANT DD(BU, 7), 7 --
in the instruction --
L, CONSTANT --,
the implicit data description attached to the Load instruction is (BU, 7).

Pseudo dds

When CONSTANT has not been defined by the programmer, the pseudo dds associated with the instruction --
L, CONSTANT --
will be (N) which is the data description assigned to the undefined symbol CONSTANT.

Types of Symbols

undefined: A symbol is undefined if it is used in an address field but has never appeared in a name field to specify either a datum or an instruction.

unused: A symbol is unused if it has appeared in a name field referring to a datum or to an instruction but has not been referenced by an address field.

multiple-defined without contradiction: This type of symbol is here defined by an example:

```
RAT     SYN, 5
RAT     SYN(BU, 32), 5
```

Note the additional specifications (BU, 32) on the symbol, RAT, does not contradict any previous specifications of the symbol.

multiple-defined with contradictions: This type of symbol is also best defined by an example:

```
DOG     LX, 5, CAT
DOG     SX, 5, MONKEY
```

In this case a contradictory condition exists as DOG is referencing two different instructions.

circularly defined symbol: This type of symbol is defined by an example:

```
A     SYN, B + 2
B     SYN, A
```

contagious error: A contagious error occurs wherever a programmer symbol is defined in terms of another programmer symbol which has been erroneously defined in one of the four ways described above.

Unit

This refers to the intermediate expanded instruction unit record built up by STRAP II for an instruction.

VFL

This stands for variable field length.

vle

This stands for variable length entry, in particular the variable portion of a symbol table entry.

X-Datum

See bit style number.

An index specification, if used, must be the last entry in the field.

Correct: B, JOE + 1.32 ($3)

Incorrect: B, JOE ($3) + 1.32

A subscript, if used, must immediately follow the symbol to which it refers.

Correct: BB, JACK(1, 2) + .32, BILL

Incorrect: BB, JACK + .32(1, 2), BILL

A gp must follow all other (type) information in the field in which the gp appears. Therefore an index and a gp cannot both be specified in the same field.

Illegal: L, JOE ($2) (.31) 3

Legal: L, JOE (.63) 2 (.31) 3

Legal: L, JOE ($2), 0 (.31) 3

A general parenthetical field integer entry cannot be the first entry in a field.

The .n in the gp format must be absolute; the binary pattern may be absolute or symbolic.

A gp cannot be specified inside another set of parentheses except in the DD case where a gp can appear in the data description.

STRAP II will make a symbol table entry for all symbolic statement names, even if they appear in an instruction where a name is not normally meaningful, i.e., SKIP, TAIL, etc. .

Implied multiplication is not allowed in field arithmetic.

The following remarks concern the radix under which operands in an 'address' field of a numeric DD are processed:

1. If a radix is specified in the leading entry mode before the DD mnemonic and no radix is specified before the first operand in the field, then all the operands in the field will be processed under this radix specified in the leading entry mode.
2. If a radix is specified before the first operand in the 'address' field, this radix overrules the leading entry mode, if any, for the current field only and all operands of the current field will be processed under the radix specified before the first operand.
3. If there is no leading entry mode and there is no radix specified before the first operand in an 'address' field then all the operands in the field will be processed under the STRAP II assigned radix of 10.

When a general parenthetical integer entry is specified on the field length or on the byte size of a DD, the gp will be OR'ed into each of the data fields.

Example: DD(BU, 32(.6)7), 0, 1, 2, 3, 4

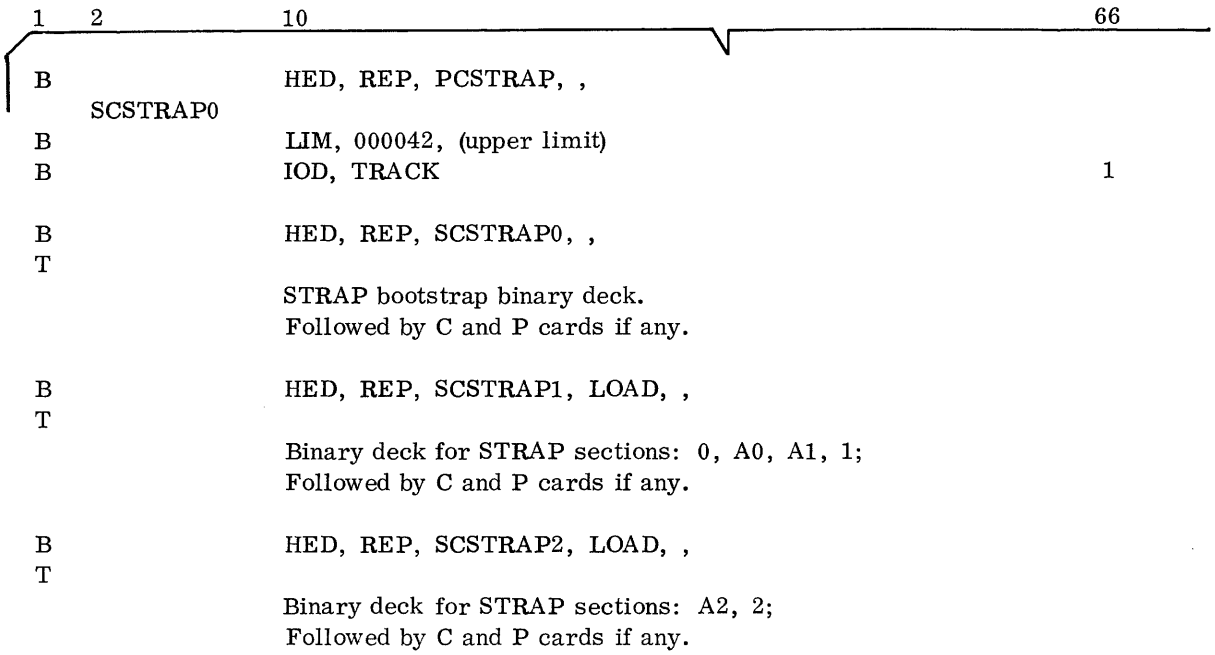
The gp will be OR'ed into each of the data fields.

The dds associated with a particular system symbol can be specified in an instruction with a P-mode.

Example: L(P, $LZC), ALPHA

The dds assembled in the load instruction will be (BU, 7, 8).

## APPENDIX G - COMPOSITION OF STRAP II's BINARY DECK



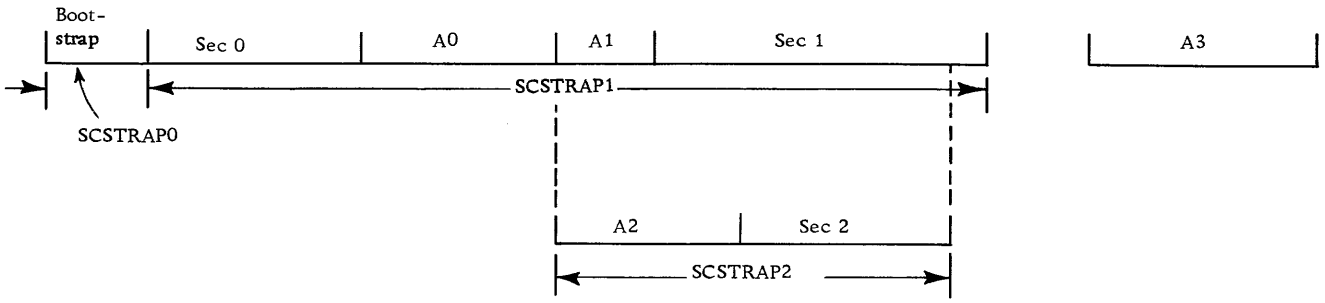



Figure 30. Memory Map of STRAP II

CUT ALONG LINE

## COMMENT SHEET

## IBM 7030 ASSEMBLY PROGRAM

PROGRAMMING SYSTEMS ANALYSIS GUIDE, FORM C22-6729

**FROM**

NAME _____

OFFICE NO. _____

FOLD

**CHECK ONE OF THE COMMENTS AND EXPLAIN IN THE SPACE PROVIDED**

FOLD

☐ SUGGESTED ADDITION (PAGE )

☐ SUGGESTED DELETION (PAGE )

☐ ERROR (PAGE )

**EXPLANATION**

FOLD

FOLD

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

FOLD ON TWO LINES, STAPLE, AND MAIL

STAPLE

C22-6729

FOLD FOLD

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY
IBM CORPORATION
P. O. BOX 390
POUGHKEEPSIE, N. Y.

ATTN: DEPT. B71

CUT ALONG LINE

FOLD FOLD

Printed in U.S.A. C22-6729

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York

STAPLE