HEWLETT-PACKARD CROSS ASSEMBLER
HEWLETT-PACKARD, CROSS ASSEMBLER

by

Khursheed Ahmed

'A dissertation submitted to the Department of Applied Mathematics, McMaster University, Hamilton, in partial fulfilment of the requirements for the degree of Master of Science.

August 1974

© KHURSHEED AHMED 1975
ABSTRACT

The Hewlett-Packard Cross Assembler is a program available on CDC-6400 computer and is used to assemble HP-2000 series assembly language programs. The binary code for the assembled programs can be punched on cards for execution on HP-2100A computer.

This report describes the design and working of the Cross-Assembler. A program listing (in PASCAL) and a few sample runs are also included.
ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks to Dr. W.H. Fleming for his keen supervision and constant encouragement during the course of this project. He also wishes to thank Dr. N.L. Solnsteff and Dr. D. Kenworthy for many useful suggestions and helpful criticism.

Very special thanks are due to Miss Katie Sajder for her patience in typing this thesis.
CONTENTS

INTRODUCTION

1.1 Machine Language 1
1.2 Assembly Language 1
1.3 An Overview of the Cross-Assembler 3
1.4 HP-2100A Computer 7

Chapter I FIRST PASS 12

1.1 General Description 12
1.2 Symbol Table 16
1.3 File S 18
1.4 Instruction Processing in PASSONE 19

Chapter II SECOND PASS 31

2.1 General Description 31
2.2 Instruction Processing in PASSTWO 36
2.3 Literal Constants 50

Chapter III BINARY OUTPUT 53

3.1 Absolute Binary Output 54
3.2 Relocatable Binary Output 56

CONCLUSIONS 60
APPENDICES

A - Instructions for Using the Cross-Assembler 62
B - Alphabetic List of Assembler Instructions 65
C - Assembler Instructions (by Groups) 70
D - Error Messages 73
E - Absolute Binary Format 76
F - Relocatable Binary Format 77
G - Program Listing of the Cross-Assembler 82
H - Sample Runs under the Cross-Assembler 124

REFERENCES 140
LIST OF FIGURES

Fig. 1.1  General Flow of Control
         (of the Cross-Assembler)  5

Fig. 1.2  HP Word Format  8

Fig. 1.1  Flow chart for PASSONE  13

Fig. 2.1  Flow chart for PASSTWO  33

Fig. 2.2(a) Multiple Instructions
          (Shift-Rotate Group)  38

Fig. 2.2(b) Multiple Instructions
          (Alter-Skip Group)  39
INTRODUCTION

1.1 Machine Language

In the early days of computers a program had to be coded by hand, as a sequence of binary numbers, constituting the coded machine instructions and data for the program. Such a program, called a machine-language program, was very cumbersome to write. The programmer had to specify the correct machine code for the instructions, assign locations (or addresses) to each instruction or data element in the computer's memory, and then use these addresses in the instructions.

Apart from the difficulty of keeping track of a lot of numbers, it was very difficult to make changes in a machine language program.

1.2 Assembly Language

Attempts were made to design better ways of writing computer programs. This led to the development of Assembly Languages \(^1\) and, subsequently, to high-level languages \(^2\).

The assembly language approach is somewhat similar to the machine language approach but makes it much easier to program. In an assembly language \(^1\), all the operation codes are mnemonics rather than numeric, which are easier to remember and recognize. The addresses of memory locations can be assigned symbolic names which can then be used in the

\(^1\) High-Level Languages

\(^2\) High-Level Languages
machine instructions to refer to a particular location in memory. Also, the programmer need not specify the locations where the instructions would be stored in the memory, rather, he assumes the starting address as location zero (or some other address) and develops the program relative to this address. Extra pseudo-instructions are included to provide additional programming convenience and defining constant data.

Thus, an assembly language program essentially incorporates the same logic and style as a machine language program but relieves the programmer of a lot of tedious bookkeeping.

Typically, an assembly language statement consists of three fields, viz., the location field, the operation field and the operand field. These are separated by blank characters. Comments can usually be included after the operand field or on separate cards specially coded for this purpose.

The location field may be empty or contain a unique label name. This label is then associated with the address at which the instruction will be stored and can be used to refer to this location.

The operation field consists of a short mnemonic for the instruction or the pseudo-instruction and is not optional.

The operand field generally contains an arithmetic expression, consisting of symbolic names (defined as labels to some instruction), and numbers. The nature of the operand
depends on the nature of the instruction; in some cases an operand may not be present at all. For most instructions, the operand will evaluate to a number that corresponds to a location in the computer's memory.

The assembly language may be more convenient for the programmer, however, the only language that the computer-hardware can understand to perform its function is its machine language. We, therefore, need a mechanism to translate an assembly language program into the machine language. A program which performs this function is called an Assembler.

1.3 An Overview of the Cross-Assembler

This report describes an assembler program for the Hewlett-Packard* 2000 series assembly language programs. It is called a Cross-Assembler because it is not implemented on an HP computer but on a different machine, viz., the CDC-6400 computer. It has been written in the programming language PASCAL.

The motivation behind this project was to develop a support system for the HP computer of the Department of Applied Mathematics at McMaster University, to take some of the work away from it with the aid of the CDC-6400 computer on the campus. It can be used for debugging HP assembly language programs as well as for producing punched binary output which

* We shall use the abbreviation HP for Hewlett-Packard 2100A Computer.
can be loaded into the HP Computer for execution. Also, it produces an improved error diagnostic over the standard HP Assembler.

The instructions and control cards necessary for using the Cross-Assembler are described in Appendix A.

A general flow of control of the Cross-Assembler is shown in Figure 1.1. Like most assemblers, it goes through two passes. The first pass is used to produce a symbol table of all the symbolic names appearing in the program and to assign a value of the location counter to these symbols. The second pass re-examines the entire program and produces the machine code in a binary form; it uses the symbol table to substitute addresses for the symbolic names.

**FIRST PASS**

In the first pass each line of the source program is read from cards, one at a time, and a value of the location counter is associated with each instruction. The mnemonic for the instruction is extracted and decoded to determine the nature of the instruction. If a symbol appears in the label

* Relocatable programs are assembled assuming the starting location as address zero. When loading such a program in the HP computer, the Relocatable Loader determines the actual starting location in the memory and adds this base address to all the affected operands. In an absolute program the actual program origin is specified by the programmer.
START

INITIALIZE VARIABLES

READ USER'S CARD
PRINT TITLE PAGE
PUNCH HEADER CARD

READ AND INTERPRET CONTROL CARD

ERRORS?
YES

SKIP PROGRAM (UPTO THE FIRST END STATEMENT)

NO

PASSONE

SYMBOL TABLE REQ'D?
YES

PRINT SORTED SYMBOL TABLE

NO

PASSTWO

FATAL ERRORS?
YES

PUNCH ERROR CARD

NO

MORE PROGRAMS?
YES

PRINT BINARY IN PROPER FORMAT

NO

END

GENERAL FLOW OF CONTROL
Fig. 1.1
field (i.e. the location field) it is extracted and entered in the symbol table (using hash-coding), along with the value of the location counter. Most machine instructions require one word of storage, while a few require two words of storage. The location counter is incremented by the appropriate amount.

Some pseudo-instructions, those providing program-linkage information or dealing only with storage-allocation, are completely processed in the first pass. All the other pseudo-instructions are partially processed to determine the amount of storage they might require and this storage is set aside (by incrementing the location counter).

Operand field is generally not processed in the first pass except for a few pseudo instructions.

All the instructions and the decoded information regarding these instructions is saved on a file so that it can be used in the second pass.

SECOND PASS

The actual machine-code is generated in the second pass, with the help of the information gathered in the first pass.

Each instruction is re-examined and the binary-code for the instruction substituted for the mnemonic. The operand field is scanned to evaluate the symbolic expression; the symbols are replaced by the values of their location counter from
the symbol table; which results in the operand address. The

data generating pseudo-instructions are also completely pro-
cessed in this pass and the binary code they produce is stored
in the location that were reserved for this purpose in the first
pass.

If binary output is required then the assembled binary
code is stored on a file. On completion of the second pass,
the assembled code can be printed out as a set of binary rec-
ords or punched on cards in the format acceptable by the HP
Computer.

In the later chapters, we shall elaborate upon this
brief description of the Cross-Assembler. The next section
gives a brief introduction to the HP Computer.

1.4 HENLETT-PACKARD 2100-A COMPUTER

In this section we shall describe, very briefly, the
basic features of the HP-2100 A Computer that concern an
assembly language programmer. A detailed description can be
found in the Reference Manual \(^4\) for the computer.

HP Words

The length of an HP word is 16-bits, numbered from
0 through 15, in order of increasing significance, as shown
in Figure 1.2. Two words can be combined together to form
a double-word (32-bits) for use as a floating-point number
or an extended-precision integer. When addressing double-
Sign bit

Least significant data bit

Binary point

Fig. I.2
HP WORD FORMAT
word in memory, the address refers to the least significant word location; the next higher location contains the most significant word. Double-precision numbers are similarly represented using three HP words.

For single-word data, the range of representable numbers is -32768 to +32767 (decimal), or -100000 to +77777 (octal). A real number, represented as double-word, must lie in the range $10^{-38}$ to $10^{+38}$ (approximately).

Pages of Memory

The 2100 A Computer can be equipped with any one of the six memory configurations, viz., 4K, 8K, 12K, 16K, 24K and 32K, ($K = 1024$ decimal words). The memory is logically divided into pages of 1024 words each (2000 in octal notation). The first page (addresses 00000 to 01777) is called the Base Page or the Zero Page. Any other page currently being referenced is called the Current Page. Memory reference instructions include a bit (bit 10) to specify the base page, or the current page (0 to 1 respectively).

Direct and Indirect Addressing

A memory reference instruction can address a location directly or indirectly. If it has a 0 in bit - 15 then the specified address is used (direct-addressing); if bit - 15 is 1 then the content of the addressed location is used as the address (indirect addressing).
Registers

There are two accumulators, viz., the A and B registers. They may also be addressed as memory locations 00000 and 00001 respectively. These registers are used to hold the results of arithmetic and logical operations, load and store data and perform comparisons. An OVERFLOW register, consisting of one bit, is set to 1 when an overflow in the register A or B occurs. The overflow bit can be cleared, set or tested.

There are also a number of other registers but we shall not be directly concerned with these. A complete description can be found in reference 4.

HP Assembly Language

There are 80 machine instructions for the HP-2100 A Computer. They can be divided into five groups:

1. Memory Reference Instructions
2. Register Reference Instructions
3. Input/Output Instructions
4. Extended Arithmetic Memory Reference Instructions
5. Extended Arithmetic Register Reference Instructions

The instructions in the first three groups are single word instructions, while those in the last two groups produce two words of machine code.

Up to eight instructions of Register Reference group can be packed into a single word, according to certain rules. We shall call such an instruction a Multiple Instruction.
In addition, the HP Assembly language includes 26 pseudo-instructions. These are used to generate constant data, provide program linkage, specify nature and starting address of assembly and provide other programming conveniences.

An alphabetic list of all the machine and pseudo-instructions and their description is included in Appendix B. A list of these instructions arranged by groups appears in Appendix C. For a detailed description of these instructions, the reader should refer to HP Assembler Manual5).
CHAPTER I
FIRST PASS

1.1 General Description

As indicated in the Introduction, the Cross-Assembler consists of two passes. The First Pass is used to determine the storage allocation of each instruction and generate a table of symbols. The actual machine-code and program listing is generated in the Second Pass. Punched output of the binary code, if required, is produced by special routines following the Second Pass.

In this chapter we shall describe the First Pass procedure PASSONE in detail. A flow-chart for PASSONE appears in Figure 1.1. This flow-chart gives only an approximate, over-all sort of picture and does not show all the details. The reader should refer to the appropriate section of this chapter or the program listing, appearing in Appendix G, for all the details. The nature of the program variables will be explained as they enter into the discussion; in some cases they are also explained in the program listing.

The main program first reads and decodes the control cards (described in Appendix A) and initializes some of the global variables. It then calls the procedure PASSONE.
FIG. 1.1
FLOW-CHART FOR PASS ONE
Extended Arith. Instr.

73 ≤ OPN ≤ 81

NO

Pseudo operation

ABS,DEF

ASC

BSS

COM

DECO,DEX,OCT

ENT

EXT

EQU

Store label

LC+LC+1

LC+LC+N

Store label

SCANOPERAND (N)

Store common block
names and sizes

LC+LC+2

OPN ≥ 78

NO

YES

Store FLBL[OPN] as an EXT symbol

PASS I

Fig. 1.1
(cont.)
IFN

OPTION='N'

YES

NO

Skip all instructions until XIF. Store them as comments.

IFZ

OPTION='Z'

YES

NO

Extract and store Program name

ORG

SCANOPERAND(LC)

PAGEFLAG=TRUE

ORR

PAGEFLAG=TRUE

ORB

PAGEFLAG=FALSE

REP

SCANOPERAND(REPN)

REPLAG=TRUE

HED, SKP, SPC, SUP, UNL, UNS

XIF

Terminate IFN or IFZ pseudo

END

Extract transfer address

Check validity of ENT symbols

End of Pass One

Fig. 1.1
(cont)
This procedure reads the source program (in HP Assembly language) one card at a time, and decodes it partially. It assigns a value of the location counter to each instruction and extracts any symbol appearing in the label field. Normally, the symbols are treated as 'program relocatable,' i.e., if the program origin is moved up (in the memory) by N locations then this number must be added to the location counter of each of the symbols. However, the programmer may specify a section of the program to be assembled on the Base page (to be discussed later), in which case the symbols are Base-page relocatable. Special non-relocatable, (i.e. absolute symbols, external symbols and common-block specificators may also be defined by means of special pseudo-instruction; as we shall see in Section 1.4.)

1.2 Symbol Table

All the symbols appearing in the program, the numeric value of the corresponding counter and their relocatability information is stored in a symbol table, SYMTAB. This table is an array of Pascal records, each of which has the following structure:

SYMTAB [I].NAME (type:alfa) contains the symbol name
SYMTAB [I].LOC (type:integer) contains the value of the appropriate counter
SYMTAB[I].TYP (type:character) contains the relocatability code for the symbol, viz.,
- R, program relocatable
- B, base-page relocatable
- C, common symbol
- X, external symbol
- blank, absolute.

Entries to symbol table are made by using a hash-code addressing technique. The following scheme is used:

\[ H = ((\text{ABS} (\text{ORD}(\text{NAME}))) \mod 4096) \mod \text{SZ} \]

where \( H \) = hash-address

\( \text{NAME} = \) Symbol, as an alpha variable

\( \text{ORD}(\text{NAME}) = \) Internal representation of \( \text{NAME} \) on CDC

as an integer (left-justified),

\( \text{SZ} = \) size of symbol table (an odd number).

All the \( \text{NAME} \) fields of SYMTAB are initially blank-filled. As each symbol is encountered, its hash-address \( H \) is calculated and if the location \( H \) is free, the symbol is stored there. If, however, the location is occupied by another symbol then the addresses \( H+1, H+3, H+6, H+10, \ldots \)
\( H+n(n+1)/2 \) are searched (spaced to reduce clustering) until a free location is found and the symbol is stored at that location. Doubly defined symbols are flagged as errors.

The operation field is extracted and the mnemonic
is checked for validity in the list of legal mnemonics, OPTABLE, and its identification number is determined. Illegal operations are flagged as errors.

1.3 File S

All the decoded information as well as the actual card-image is stored on a file S, which is a temporary disc file, so that the information may be used in the Second Pass. The file S consists of Pascal records; each record has four fields, namely,

- LN (integer) - Serial number of the card in the source deck.
- LOCFLD (integer) - Value of the location counter
- OPFLD (integer) - Identification no. of op.
- LINE (array of alfa) - Compressed form of the actual card image.

A zero value of OPFLD is stored for the comment cards, as well as for the statements with an illegal operation field, which are then treated as a comment cards.

The operand field is usually not processed in PASSONE, except for a few pseudo-instructions; these will be discussed later-on in this chapter. Since the card-image is stored on file S, it can be used for scanning the operand field in the Second Pass.
No program listing is produced in PASSONE; instead, only those lines that contain errors, along with the error messages, are printed-out. A complete program listing will be produced in PASSTWO.

1.4 Instruction Processing in PASSONE

One source line is processed in each cycle of PASSONE, until the END pseudo or end-of-file is encountered. The source line is read into a character array INTEXT and stored in a compressed form on file S (as $S \times LINE$). If an asterisk (*) appears in the first column then it is treated as a comment card. If the first column is blank then there is no label field and the alfa variable LBL is set to EMPTY (i.e. 10 blanks packed to a word), otherwise the label name* is extracted and stored in SYMTAB.

The operation field is also extracted and the mnemonic (stored in the alfa variable OP) is checked in a table of operations (OPTABLE) and its identification number OPN is extracted. Illegal operations are flagged as errors. A zero value of OPN is stored both for comment cards and illegal operations, which will then be treated as comments and ignored.

* ten characters packed to an alfa variable
  i.e. a CDC word

* The label must start with an alphabetic character
  or a period(.) and may contain up to five alphabetic characters and periods (.).


In an absolute program an ORG statement must appear before code-generating instructions. Similarly, in a relocatable program a NAM statement must appear before code-generating instructions.

A boolean variable FLAG (initially TRUE) is set to FALSE when the appropriate pseudo (NAM or ORG) is encountered. While FLAG is true, each OP is tested to make sure that it is not a machine instruction or a code-generating pseudo. Otherwise, an error message is printed out but the assembly process is not terminated.

The specific action taken in PASSONE depends on the nature of the instruction. They can be described by dividing them into the following classes:

**Machine Instructions**

If the operation field contains a machine instruction (1≤OPN≤73) then the location counter is incremented by 1. However, if the previous instruction was the pseudo-instruction REP n (indicated by a TRUE value of the boolean variable REPFLAG) then the location counter is incremented by n and REPFLAG is set to FALSE. No further action is taken at this stage except the routine actions (e.g. storing the label and card-image etc.) described earlier.
Multiple Instruction

If the operation field contains a number of operation mnemonics, separated by commas, then the value of OPN for each of these mnemonics is stored as a negative number (=-OPN), to distinguish it from an ordinary or single instruction. Thus a multiple-instruction will be stored on several records of the S file, each record representing a mnemonic of the multiple-instruction. An extra record with OPN=0 is stored to indicate the end of a multiple instruction. In PASSTWO this string of records will be combined to generate one word of machine code. The location counter is incremented by 1 (or by n if the preceding instruction was a pseudo REP n).

Extended Arithmetic Instructions

The eight instructions in this group (74≤OPN≤81) produce two words of code each, so that the location counter is incremented by 2. The first four instructions (viz. MPY, DIV, DLD, and DST) are assembled such that the machine code is stored in the first word and the operand address in the second word (to be done in PASSTWO). However, the remaining four instructions (viz FAD, FSB, FMP and FDV) are assembled as subroutine calls to the external labels .FAD, .FSB, .FMP and .FDV, respectively. These labels are, therefore, stored in SYMTAB as external symbols when the corresponding
instructions are encountered. In PASSTWO, when machine code will be generated, the first word will contain a JSB call to this label and the second word will contain the operand address.

Pseudo-Instructions

There are 27 pseudo-instructions for the HP Assembly language (see Appendices B and C). Some of these pseudos result in machine code, while others provide information about program origin, program linkage, numeric and character data, listing control etc. We shall describe their PASSONE actions by classifying them into sub-groups.

For all the pseudo instructions, as for any other instruction, the first action is to store the value of current location counter, the value of OPN (instruction number) and the card-image on the file S.

1) Assembler Control

NAM: This pseudo is illegal in an absolute program and must appear before the first valid opcode in a relocatable program. Both these conditions are checked out. The operand field is scanned to check and extract the program name and type (a number indicating if it is a main program, a sub-routine, etc.) and stored in PROG (alpha variable) and NMN (integer variable) respectively, to be
used in preparing the binary output (chapter III). Both these are optional parameters and if they are not specified, they are set to PROG = EMPT and NUN = 0.

**ORG:** An ORG statement specifies the origin of an absolute program or origin of a segment of an absolute or relocatable program. Thus an absolute program must begin with an ORG pseudo, and a relocatable program (which must begin with a NAM pseudo) may contain ORG pseudos within the program. These conditions are checked first.

The operand address is evaluated and the program location counter (LC) is set to this value. The succeeding code will be assembled on the current page starting at this location. Boolean variable PAGEFLAG is set TRUE. (When PAGEFLAG is TRUE, code is assembled on the current page, otherwise it is assembled on the Base page).

**ORB:** The ORB statement defines the Base page portion of a relocatable program and is illegal in an absolute program. A check is made to ensure this condition. The base page assembly is initiated by setting PAGEFLAG to FALSE. The effective location counter is now BPC (Base page counter).

**ORR:** This pseudo reverts assembly to current page (set PAGEFLAG = TRUE) starting at the value of LC that
existed when the first ORG or ORB instruction was encountered.

REP: Any label present in the location field is stored in the symbol table. The operand is evaluated and stored in REPN (integer) and REPFLAG is set TRUE. The succeeding machine instruction will be repeated REPN times.

IFN, IFZ: In the Assembler control card (see Appendix A), the programmer may specify N or Z option. If N-option is chosen then all the instructions between an IFZ and XIF pseudo are ignored while those between an IFN and an XIF pseudo are assembled; and vice-versa for the Z-option.

PASSONE first checks to insure that this pseudo, IFN (or IFZ), does not follow another IFN or IFZ (the boolean variable NZFLAG should be FALSE), otherwise an error message is printed out. The Assembler control option (N or Z) is stored in the character variable OPTION. If we encounter an IFN pseudo with an N-option, on an IFZ pseudo with a Z-option, then we simply proceed normally without any specific action. However, if a conflict occurs (eg. IFN with Z-option, or IFZ with N-option or either. IFN or IFZ without any option specified) then all the following instructions until an XIF are treated as comments and not assembled.
A special routine, GETXIF, is called to perform this task. It stores the instructions on file S with OPFLD=O (i.e., as comments) and also checks for an end-of-file condition to avoid reading past e.o.f. in case of a missing XIF.

Object-program Linkage

COM: This pseudo defines the names and sizes of the elements of the common block. It is invalid in an absolute program. For a relocatable program, the symbolic names appearing in the operand field are extracted and stored in SYMTAB as common relocatable symbols, along with the size of the array. The size of the block is specified within brackets following the symbol and if left out, it is assumed to be one HP word long.

ENT: This pseudo specifies the entry points or entry labels in the program. The symbols appearing in the operand are extracted and stored in a special table, called ENT (actually it is a table of two records, the entry symbol and its address). At the end of the PASSONE all the entry symbols stored in this table are checked for validity (they must appear as labels to some machine instruction and should not be external or common relocatable symbols) and their addresses determined from SYMTAB and stored.
in the second record of ENT table.

EXT: This pseudo defines external symbols, and is illegal for an absolute program. For a relocatable program, the symbols appearing in the operand field are extracted and stored in SYMTAB as externals.

Address and Symbol Definition

DEF, ABS: In both cases, the PASSONE action consists of storing the label (if present) and incrementing the location counter by 1. The rest will be done in PASSTWO.

EQU: This pseudo assigns a specific value of location counter to a label, but does not generate machine code (so that the location counter is not incremented). The PASSONE action consists of extracting the label from the location field and entering it in SYMTAB at the address specified as a number (or expression) in the operand field. Symbols appearing in operand expressions must be previously defined so that their addresses can be determined from the SYMTAB; otherwise an error message is printed out.

Constant Definition

ASC: This pseudo enables an ASCII string of characters to be stored in machine code. A number n preceding the string (in the operand field) determines the length of the string (2n characters), to be packed
in n consecutive words. Since, in the first pass we are mainly concerned with the storage allocation, we simply determine n and increment the location counter by this amount. (n must be such that 1 \leq n \leq 29).

The string will be processed in PASSTWO.

OCT, DEC, DEX: These pseudo-operations generate constant data specified as octal integers, decimal integers or reals and double precision integers or reals, respectively. The operand field in each case may contain one or several constants (separated by commas). Each constant in an OCT pseudo generates one word of code; each constant in a DEX pseudo generates three words of code and a constant in a DEC may generate one or two words of code, according as it is an integer or a real number.

In PASSONE all we are interested in is to determine the storage allocation of each instruction so we do not actually evaluate the constants in the operand field, rather we scan them and look for 'E' and 'E' characters to decide whether a number is integer or real, and set aside the appropriate storage as required by the specific pseudo. All these details are taken care of by a special routine called COUNTCONST.
Storage Allocation

BSS: This pseudo reserves a block of words of a specified number as a work area. Any label appearing in the label field is entered in SYMTAB; the location counter is incremented by the number found in the operand field.

Assembly Listing Control

This group of seven pseudo instructions specify listing control of the final output. Since the final listing will be produced in PASSTWO, these will be dealt with in PASSTWO. Some of these pseudos will not be included in PASSTWO listing (eg. SPC, SKP, etc), so that PASSONE stores their line number (ST.LN) as -1; others are simply ignored by PASSONE.

End Pseudo

END: This should be the last statement of a program. If the END pseudo is missing and an end-of-file is encountered then PASSONE generates an END pseudo but prints out a warning. The operand field of the END pseudo may optionally contain a transfer symbol that also appears as an instruction label in the program. When the program will be loaded for execution then execution will begin by a JSB to this point. PASSONE
scans the operand field and if a symbolic name is found, it is searched in SYMTAB and its location counter and relocatability is determined. This information will be needed by the punch routines and is stored in a special integer array TRNSFR, as follows:

If the operand contains a valid label:

- TRNSFR [1] = 0 (if label is program relocatable)
- TRNSFR [2] = 1 (if label is base-page relocatable)

otherwise, all the three elements are set to 0.

This completes the discussion of the actions taken by PASSONE on each type of instruction and pseudo-instruction. At this point in assembly, all the symbols have been entered in SYMTAB and the storage requirement of each instruction has been established. All the decoded information and the card images have been stored on file S.

The final value of the program location counter (LC) is stored in an integer variable ENDL.C. In PASSTWO, all the literals used in PASSTWO will be stored starting at this location.

Before leaving PASSONE, a check is made to ensure
that all the entry symbols appear as labels to some instruction and that they do not also appear as common or external symbols. If any discrepancy is noted then an appropriate message is printed out.

The last action in PASSONE is to print out the total number of errors, if any, detected in PASSONE.

The control now returns to the main program. If a symbol table has been requested then a sorted symbol table, in the order in which symbols appear in the program, is printed out. The second pass is then invoked by a call to PASSTWO; a detailed description of which appears in Chapter II.
CHAPTER II

SECOND PASS

The first pass assigned a value of the location counter and allocated storage to each statement in the program. It also produced a symbol table of all the symbols used in the program, their location and their relocatability. All the decoded information, as well as the compressed card-images were stored on a temporary disc file S.

2.1 General Description

In this pass, the actual machine-code and the assembled listing of the program will be produced. A flow-chart of the second pass procedure PASSTWO appears in Figure 2.1.

If the user has requested binary output, then it will be stored on two disc files R and B. The file R contains the current-page portion (or program relocatable portion) of the code, while the file B contains the base-page portion of the code. Both these files have similar structures; each record of the file consists of the following four fields:

- LOC (integer) value of the location counter
- WORD (integer) the machine code
<table>
<thead>
<tr>
<th>TYP (char)</th>
<th>relocatability of the operand, viz.</th>
</tr>
</thead>
<tbody>
<tr>
<td>=R</td>
<td>program relocatable,</td>
</tr>
<tr>
<td>=B</td>
<td>base page relocatable</td>
</tr>
<tr>
<td>=C</td>
<td>common relocatable</td>
</tr>
<tr>
<td>=X</td>
<td>external</td>
</tr>
<tr>
<td>=blank</td>
<td>absolute</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MREF (integer)</th>
<th>contains the operand address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>if the operand is a memory-</td>
</tr>
<tr>
<td></td>
<td>reference instruction; otherwise</td>
</tr>
<tr>
<td>MREF = -1</td>
<td></td>
</tr>
</tbody>
</table>

An absolute program does not have any base-page relocatable coding so that only the file R is used in such a case.

Generating the machine-code involves going through all the statements of the program again. If the operation appears alone, without any operand field, then the machine code is simply the op-code for the operation. However, if an operand is expected then the operand field must be scanned. Any symbols found are replaced by the values of their location counters from the symbol table. Some operand fields may involve literal constants (see the last section of this chapter); these are stored at the end of the program after the last opcode, and their addresses determined. The operand address is then appended with the opcode of the instruction to give the machine-code.
Fig. 2.1 (cont)
STORE LITERALS after the last OPCODE

End of Passtwo

Fig. 2.1 (cont.)
One statement of the assembly program is processed in each cycle of PASSTWO. After initializing some of the variables, the file S is reset and one record of S is read per cycle. The value of location counter (St. LOCFLD) is stored in LC, that of the operation number (St. OPFLD) is stored in OPN and the compressed card image (St. LINE) is expanded and stored into the character array INTEXT, (except for comment cards), to allow a character by character scan.

2.2 Instruction Processing in PASSTWO

The value of operation field determines the nature of the instruction and further processing depends upon what class it belongs to. We shall discuss the action on each class of instruction. Unless specified, the card number, the location counter, the machine-code and the card-image for the statement is printed out.

Comment Card

A zero value of the OPN indicates a comment card or an unrecognizable instruction (flagged as an error in PASSONE), to be treated as a comment card. No processing is done and the card number (St. LN) and the card-image (St. LINE) is printed out.
Multiple Instruction

If OPN has a negative value, it signals the start of a multiple instruction, spread over the next few S-records, terminated by a zero value of OPN. Such an instruction generates one-word opcode.

A typical multiple-instruction may be

CMB, INB

This will be found on the file S on two consecutive records with OPN having the values -69 and -72 respectively, followed by a third record with zero value of OPN. A special routine, MULTINST, is called to handle the multiple instruction. However, some description of multiple instructions will be necessary to understand its processing.

The multiple instructions fall under two groups:

a) shift-rotate group
b) alter-skip group, as shown in Fig. 2.2(a) and Fig. 2.2(b) respectively. Some mnemonics, e.g. CLE, SLA and SLB are common to both the groups but have different binary codes. A given multiple instruction may refer to either A-register or B-register but not both. In addition there is a certain order, relative to each other, in which the mnemonics may appear in a multiple-instruction. These restrictions are shown in Figs. (2.2 a & b). Only one mnemonic may appear from each pair of square-brackets, in the order shown. If only one mnemonic appears, it is treated as a single machine instruction (see the next section).
A-Register type

\[
\begin{align*}
\{ & \text{ALS} \\
& \text{ARS} \\
& \text{RAL} \\
& \text{RAR} \\
& \text{ALR} \\
& \text{ALF} \\
& \text{ERA} \\
& \text{ELA} \}
\end{align*}
\]

\[
\begin{align*}
& \{ \text{CLE} \} \\
& \{ \text{SLA} \}
\end{align*}
\]

B-Register type

\[
\begin{align*}
\{ & \text{BLS} \\
& \text{BRS} \\
& \text{RBL} \\
& \text{RBR} \\
& \text{BLR} \\
& \text{BLF} \\
& \text{ERB} \\
& \text{ELB} \}
\end{align*}
\]

\[
\begin{align*}
& \{ \text{CLE} \} \\
& \{ \text{SLB} \}
\end{align*}
\]

Fig. 2.2'(a)

MULTIPLE INSTRUCTIONS (Shift-Rotate Group)
A-Register type

\[
\begin{align*}
\text{CLA} & \quad \{\text{,SEZ}\} & \quad \text{CLE} & \quad \{\text{,SSA}\} & \quad \{\text{,SLA}\} & \quad \{\text{,INA}\} & \quad \{\text{,SZA}\} & \quad \{\text{,RSS}\} \\
\text{CMA} & \quad \{\text{,SEZ}\} & \quad \text{CME} & \quad \{\text{,SSA}\} & \quad \{\text{,SLA}\} & \quad \{\text{,INA}\} & \quad \{\text{,SZA}\} & \quad \{\text{,RSS}\} \\
\text{CCA} & \quad \{\text{,SEZ}\} & \quad \text{CCE} & \quad \{\text{,SSA}\} & \quad \{\text{,SLA}\} & \quad \{\text{,INA}\} & \quad \{\text{,SZA}\} & \quad \{\text{,RSS}\}
\end{align*}
\]

B-Register type

\[
\begin{align*}
\text{CLB} & \quad \{\text{,SEZ}\} & \quad \text{CLE} & \quad \{\text{,SSB}\} & \quad \{\text{,SLB}\} & \quad \{\text{,INB}\} & \quad \{\text{,SXB}\} & \quad \{\text{,RSS}\} \\
\text{CMB} & \quad \{\text{,SEZ}\} & \quad \text{CME} & \quad \{\text{,SSB}\} & \quad \{\text{,SLB}\} & \quad \{\text{,INB}\} & \quad \{\text{,SXB}\} & \quad \{\text{,RSS}\} \\
\text{CCB} & \quad \{\text{,SEZ}\} & \quad \text{CCE} & \quad \{\text{,SSB}\} & \quad \{\text{,SLB}\} & \quad \{\text{,INB}\} & \quad \{\text{,SXB}\} & \quad \{\text{,RSS}\}
\end{align*}
\]

Fig. 2.2(b)

MULTIPLE INSTRUCTIONS (Alter-skip Group)
Some examples of valid multiple instructions are:
ALs, CLE, SLA, ALS
BLR, SLB
CLE, SLA, ELA
CMB, CME, INA, SZA, RSS
eetc.

We shall not discuss the usage or usefulness of a multiple-instruction here. The user should refer to the HP Assembler Manual.

As soon as a negative value of OPN is detected, the routine MULTINST is called in. It reads the records from file S and extracts the entire string of records with negative OPN values. It checks the following for each mnemonic:

(i) whether the mnemonic is alter-skip type or shift-rotate type and that it is of the same type as the previous ones (if any).

(ii) whether it refers to A-register or B-register and does not contradict any previous mnemonic in the string (if any).

(iii) that it occurs in the multiple instruction in the correct sequence relative to the previous mnemonics and does not break the sequence shown in Fig. (2.2 a & b).

If any of these conditions fail, an appropriate error message is printed out and the processing of the multiple
instruction is terminated. When all the above conditions are satisfied, then the appropriate binary code for each mnemonic is extracted from code tables OPCODE and OPCODE2 and appended to the integer variable WORD (initially zero) until the entire multiple instruction is completed. The value of the location counter (LC) and the opcode (WORD) are stored on file R or file B according to the mode of coding currently active. (If PAGEFLAG is TRUE, the code is stored on file R as program relocatable, otherwise it is stored on file B, as base-page relocatable).

Machine Instructions

A positive value of OPN in the range (1≤OPN≤73) refers to a machine instruction. They may be divided into two groups.

The first group of instructions (1≤OPN≤33) expects a non-blank operand field. Since the card image of the current statement is stored in the character array INTEXT, the operand field can be evaluated by a character-by-character scan, using the procedure SCANOPR2. First, a syntax check is made (only + and - operators may appear). Symbol names are extracted and substituted by the value of their location counters, from SYMTAB. The relocatability of the symbols

- by taking logical OR, (PASCAL terminology)
appearing in an operand field must be compatible; base-page, current-page and common-relocatable symbols cannot be inter-
mixed. An external symbol must appear alone, without any
arithmetic operation. Numbers appearing in the operand are
also evaluated. The resulting value of the operand-expression
is stored in the integer variable WORD. The relocatability
of the operand is determined by the relocatability of the
symbols in the operand expression. However, two base-page
symbols appearing with opposite signs will cancel out the
effect of adding a base address. The same holds for two
program relocatable symbols appearing with opposite signs.
Such operands are treated as absolute addresses. The
resulting operand-relocatability is stored in the character
variable TYP, according to the following code, (as mentioned
earlier):

\[
\begin{align*}
TYP &= R, \text{ program relocatable} \\
    &= B, \text{ base-page relocatable} \\
    &= C, \text{ common relocatable} \\
    &= X, \text{ external reference} \\
    &= \text{blank, absolute (non-relocatable)}
\end{align*}
\]

If the operand is an indirect-reference (ie. is
followed by ",I") then the 15th bit of WORD is set to 1
(by appending WORD with 100,000 octal).

If the operand is program-relocatable, common
relocatable or external reference then the HP-2100 A
convention requires that the bit 11 of the machine-code be set to 1 (i.e. append WORD with 2000 octal).

The punching routines (Chapter III) require that the reference to memory locations be coded differently. Thus, if 1≤OPN≤14 (memory-reference instruction) and the operand is either type R, B or C then the opcode is stored in WORD and the operand-address is stored in the integer variable MR. In all other cases MR has a value -1.

Finally, the quantities LC, WORD, TYP and MR are stored on file R or B according to the current mode of assembly (program relocatable or base-page relocatable, respectively).

The second group of machine instructions (34≤OPN≤73) do not require any operand field. For these instructions the value of opcode for the instruction is extracted (from the table OPCODE) and stored in WORD. The quantities LC (location counter), WORD (opcode), TYP (a blank character to indicate non-relocatable code), and MR (= -1, indicating no memory reference) are stored on file R or B, according to the current mode of assembly.

If the last instruction was the pseudo REP n then the code is repeatedly stored on the appropriate file, n times.
Extended-Arithmetic Instructions

Each instruction in this group (74≤OPN≤81) produces two words of machine code. The instructions can be divided into two groups.

The first group (74≤OPN≤77) consists of MPY, DIV, DLD and DST instructions. For these, the opcode is evaluated and stored in the first word (at location LC), as an absolute word. Then, the operand address and its relocatability are evaluated and stored in the second word (at location LC+1).

The second group (78≤OPN≤81) consists of FAD, FSB, FMP and FDV. These instructions generate subroutine calls to floating-point arithmetic routines. In PASSONE the special external symbols .FAD, .FSB, .FMP, .FDV were generated (only those that were to be used) and stored in SYMTAB. The appropriate symbol is now extracted back from SYMTAB and a JSB call to this symbol is generated and stored in the first word (at location LC). The operand address is evaluated and stored in the next word (location LC+1).

If the last instruction was the pseudo REP n then the two words of code are stored repeatedly, n times.

Pseudo-instructions

Once again, we shall discuss the PASSTWO actions
on the pseudo-instructions by dividing them into groups, as follows:

**Assembler Control**

**NAM:** This pseudo was completely processed in PASSONE so no further action is taken.

**ORB:** This statement initiates base-page assembly. This is signalled by setting PAGEFLAG to FALSE. (Initially, PAGEFLAG is TRUE).

**ORG, ORR:** Both these pseudos signify the start of current-page assembly, so we set PAGEFLAG to TRUE. (We need not worry about the operation field; this was processed in PASSONE).

**IFN, IFZ, XIF:** These pseudos were completely processed in PASSONE.

**REP:** The REPFLAG is set to a TRUE value and the operand is scanned to evaluate REPN, the number of times the next instruction should be repeated.

When the next machine-instruction, multiple instruction or extended-arithmetic instruction will be processed, it will be stored repeatedly REPN times, at locations LC, LC+1, LC+2, ... LC+REPN-1. The REPFLAG will then be turned FALSE so that no further instructions are repeated.

**Program Linkage**

All the three pseudos in this group, viz., COM, ENT
and EXT were completely processed in PASSONE so no further action is taken.

Address and Symbol Definition

DEF, ABS: Both these pseudos generate one word of machine-code. In both cases, the operand is evaluated and stored in WORD and subsequently stored in file R or B at location LC. In the case of ABS, a check is made to ensure that the operand is an absolute number otherwise an error message is printed out.

EQU: This pseudo does not generate a machine-code. However, the operand address is evaluated to be printed out as an octal number in the program listing.

Constant Definition

The four pseudo-instructions in this group were only partially processed in PASSONE to set aside the storage needed. The actual evaluation of the constants will be done in PASSTWO.

ASC: This pseudo appears as follows

\[
\text{ASC } n, \langle \text{string of } 2n \text{ characters} \rangle
\]

The value of \( n \) indicates the number of data words to be generated; each word to contain the ASCII code for two characters. First, we evaluate \( n \), then go into a loop of \( n \) cycles. The routine GETASC is called
in each cycle; it picks up the next pair of characters in the operand, determines their ASCII code (from the table ASCIICODE), packs this binary code in 16 bits of WORD such that the high 8-bits contain the ASCII code for the first character and the low 8-bits contain the ASCII code for the second character. The quantities LC, WORD, TYP (a blank character) and MR (= -1) are stored on the file R (or file B); LC is incremented by 1 and the next loop is performed similarly. Normally, the binary code for each WORD is printed out (in octals) except when it is suppressed by a SUP instruction, in which case all but the first WORD are suppressed. (See Listing Control Pseudos).

OCT: The operand field is scanned to evaluate each octal integer constant (separated by commas) and is stored at the next available location. All characters except digits 0 to 7 are considered as errors in the operand of OCT.

DEC: The operand field of the DEC pseudo may contain a string of decimal integers and reals. An integer is to be stored in a single HP word while a real is spread over two HP words. In PASSONE, we have determined the total number of memory words that this pseudo will generate.

A special routine GETCONST is called. It checks the syntax of the next constant in operand, determines
if the constant is real or integer and returns its value in the three variables, KIND(integer), X(real), N(integer) as follows:

KIND=0 - there was an error in the constant, N is set to zero.

KIND=1 - an integer was found whose value is returned in N.

KIND=2 - a real number was found, its value is returned in X and N, such that X contains the fraction and N contains the exponent to base 10.

An integer constant is stored in the next available word of code. However, the real number must first be converted in the HP format* as a normalized floating point number stored on two HP words. A special routine CONVERT is called to perform this conversion.

The process is repeated for each constant in the operand field.

**DEX:** The processing of DEX constants is very similar to those of DEC, except that both integers and reals are stored as normalized floating point numbers*, stored on three HP words (rather than two). Again, the procedure CONVERT is called to perform the proper conversion and normalization to the HP-format.

* for details, see p.4-19 to 4-23 of the HP Assembler Manual, reference 5.
Storage Allocation

BSS: This pseudo was completely processed in PASSTWO, so no further processing is necessary. The starting location counter (LC) and the length of the storage block (operand number) are printed out in octal format but no machine-code is generated.

Assembly Listing Control

Since the final program listing is produced in PASSTWO these pseudos concern us at this stage.

SPC: The value of N is evaluated from the operand and the routine LINES(N) is called to print N blank lines in the program listing. The statement itself is not printed out.

SKP: We simply skip to the top of next page of the output and print-out the program title (see HED pseudo). The statement is not printed.

HED: The program heading is extracted from the operand field and stored in the character array HDNG. This heading will appear on the top of each page of the program listing, unless a new HED statement is encountered, which changes the contents of HDNG.

UNL: Normally, each statement of the program is listed, along with the location counter and opcode. A UNL pseudo suppresses the program listing. We achieve
This by setting the boolean variable LSTFLAG to FALSE. Lines are printed out only if LSTFLAG is TRUE.

LST: This pseudo causes the source program listing, suppressed by a UNL, to be resumed. We simply set LSTFLAG to TRUE.

SUP: This pseudo suppresses the output of additional code lines (produced by ASC, OCT, DEC and DEX pseudos) from the program listing. A boolean variable EXTRALINE is used to identify these additional code lines. In addition, another boolean variable SUPFLAG (initially FALSE) is set TRUE. Thus additional code lines (EXTRALINE=TRUE) will be printed only when SUPFLAG is FALSE.

UNS: This pseudo reverts the effect of the SUP pseudo, i.e. resumes listing of extra lines of code. We simply set the SUPFLAG to FALSE.

End Pseudo

END: The pseudo signals that no more instructions are to be assembled. PASSTWO now goes on to perform other tasks, first of which is to store literals.

2.3 Literals

In order to understand the PASSTWO action on the literals, it is necessary to have a brief discussion of the types and meaning of literals allowed in HP Assembly language.
The operand field of certain machine instructions may define and use constant data by means of literals. A literal is specified by using an equal sign and one-character identifier. The actual literal constant is specified immediately after the identifier.

Some examples are:

LDA =B1377
FMP =F0.623
LBD =LSTART-SYM+5

(START and SYM are symbolic labels)

etc.

The literals are stored at the end of the program. If a literal is used in more than one instruction then it is stored only once and its address substituted for each reference to the literal.

During PASSTWO, when a literal is encountered, it is evaluated using the procedure SCANOPERAND and stored in a special table LITTAB, which has two elements,

LITTAB[1,1] = binary form of the 1st literal
LITTAB[1,2] = the address of the 1st literal.

The first literal is assigned the address ENDLC (last address of the program code + 1), and ENDLC is incremented by 1 so that the next literal may go in that location.

The identification code is the following:

D - a decimal integer
F - a floating point number
B - an octal integer
L - an expression consisting of symbols used in the program. Must evaluate to an absolute value.
Before storing the value of a literal in LITTAB, it is first searched (by a simple linear search) in LITTAB. If it is already present then its address (LITTAB[I, 2]) is extracted and returned as the operand address. If it is not present, then an entry is made and the counter I is updated.

A check is made to ensure that the literal used is compatible with the instruction in which it appears.

At the end of PASSTWO, we should have all the literals and their addresses stored in LITTAB. We can now generate actual machine code for these literals and store their addresses and binary values on file R.

The last action in PASSTWO is to print-out the total number of errors, if any, detected in PASSTWO.

This concludes the execution of PASSTWO and control is returned to the main program. If binary output is required then it is produced by means of special routines described in the next chapter.
CHAPTER III

BINARY OUTPUT

The machine code for the source program is produced in the Second-Pass. If the user requests binary output in the form of punched cards (B option in ASMB card*) or listed in the form of binary records (Q option), then and only then, the assembled machine code is stored on files R and B during PASSTWO. However, if a fatal error** occurs in either Pass One or Pass Two then the binary output is suppressed and a card with the word ERROR in big letters is punched out. If several programs are assembled in a single run of the Cross-Assembler then the punched output for each program is identified by a header card, with a serial number HP01, HP02, ... etc. which also appears in the output listing of each program.

In this chapter we shall describe the techniques used for producing the binary output in a format acceptable to the HP computer**. All the information to be punched is written on file (of integer) P. At the end of the run, this

* See Appendix A for control options and usage instructions.
** A list of non-fatal and fatal errors appears in Appendix D.
file is punched out by means of SCOPE control cards*. We shall describe this later.

On the HP-2100A the binary output for an absolute program and a relocatable program have completely different formats. We shall, therefore, discuss them separately.

3.1 Absolute Binary Output

For the absolute programs each machine word (16-bits) is broken up into two parts and each half-word (8-bits) is punched in a card column in the punch position 12 to 5. For example, the 16-bit binary number

1011 001 110 101 111

is broken up into two parts

10 110 011 and 10 101 111

and punched on a card in two columns in an inverted** manner as shown below:

*See Appendix A

**By inversion we mean transferring an 8-bit binary number so that when it is punched on a card-column it appears upside down. This is required by the HP convention and is achieved by a table look-up.
The absolute binary output of an assembled program is divided into records: each record of up to 40 words of code is punched on a separate card. The absolute record format is shown in the Appendix E.

Binary output in absolute format is produced and printed and/or punched out by means of a special routine called ABSBINRECORDS. This routine resets the file R which contains the absolute code, consisting of the location counter (the absolute address) and the machine-code (the content of the addressed location). It then starts reading these values in pairs, checking for continuity of the location counters (viz. LC, LC+1, LC+2, etc.). A record is completed when either 37 pairs are read, or a discontinuity in the location counter is found or the end-of-file on R is reached.

A complete record contains the following information which is stored in an integer array W.

- \( W[1] \) = Record length (left justified);
- \( W[2] \) = Absolute Load Address, i.e. the location counter for the first machine code in the record.
- \( W[3], \ldots, W[\text{LAST}-1] \) = Machine-code words,
- \( W[\text{LAST}] \) = CHECKSUM (i.e. Arithmetic sum of all the words in the record except the first and last, modulo 200000 octal)
The maximum number of words in a record (i.e. LAST) will be 40. A special routine STORE is called which breaks each (16-bit) integer of W into two half-words at the 8th bit. Each half-word is then inverted. Five such half-words are packed into a 60-bit CDC integer word and stored on the file of integers P. When P will be punched on cards (by means of SCOPE control card, see Appendix A), each CDC integer word will occupy five columns of a card, each column representing a half HP word, as required by the HP Basic Binary Loader.

3.2 Relocatable Binary Output

The relocatable punched format for the HP2100-A computer has a more involved structure than the absolute punched format. Not only do we have five different types of records to consider, but, in addition, each 16-bit HP word has to be punched so that it occupies one and one-third columns of a card, or three HP words will fill four card columns.** The problem is further complicated by the fact that a 60-bit CDC word is punched on five columns of a card.

One must, therefore, compress 15 HP-words (15 x 16 = 240 bits) into 4 CDC words (4 x 60 = 240 bits), which will then appear on 20 columns of a punched card. Any incomplete word must be left justified.

* See footnote on page 54
** See also reference 7, page 2-19.
The machine code of a relocatable program is divided into five different types of records, as follows:

**SAM record** - contains the name of the program unit, its type (main, subroutine, etc.) and information on lengths of various segments.

**ENT record** - contains up to 14 entry-point symbols and their addresses.

**EXT record** - contains up to 19 external symbols and their ID numbers.

**DBL record** - contains up to 60 words of base-page or current-page machine-codes.

**END record** - contains the transfer address (starting address for program execution).

A description of these records appears in reference 7 and a copy of the record-formats is reproduced in Appendix F for easy reference.

One record is punched on a computer card. Each relocatable program must contain a SAM record, a number of DBL records and an END record, and optionally, an ENT record and a number of EXT records.

The SAM, ENT, EXT and END records are rather straightforward in structure. Each of these records is handled by a
separate routine, viz., NAMRECORD, ENTRECORD, EXTRECORD and ENDRECORD, respectively. Each of them combines information saved on various tables, puts it together in the appropriate format (Appendix F) and stores it in the integer array W. Another routine, called PUNCHRECORD then picks-up W and performs the mechanical task of packing the 16-bit HP words into CDC words (as described earlier), which are then stored on an integer file P. We shall not go into any further details of these routines; the reader should refer to the program listing of the Cross-Assembler (Appendix G).

The DBL record is a little more complicated. There are two sets of DBL records; one for the base-page portion of the code and the other for the current-page (or the program relocatable) section of the code. They are identical except for one-bit in the second word (see Appendix F).

If the machine instruction does not refer to a memory location then one machine-code word is stored on an HP word.

Memory reference instructions are formatted so that they are stored on two words each. The first one contains the binary code for the instruction and the second contains the memory address.

Upto five consecutive instructions are grouped together and an extra word, appearing before the group, contains 3-bit relocatable codes (packed-together) for each
instruction in the group. The routine DLBRECORDS produces the DLBRECORDS in this format (Appendix F).

All the information to be punched is written on an integer file P. At the end of the Assembler run, the file is disposed to the punch device by the SCOPE control command.

DISPOSE (P, P8)

The file P may in general contain punched output for a number of programs, absolute and binary. However, as mentioned earlier, these can be sorted out by means of the identifying header-card which appears at the top of each punched deck.

If the user specifies a Q-option on the ASMB control card then the Cross-Assembler prints out the binary output as a set of records. The records are written in octal format and appear with the identifying information. An example of such a listing appears in Appendix H.

* This option is not available on the standard HP Assembler, supplied by the manufacturers.
CONCLUSIONS

The author feels that he has achieved the basic goal set out for this project. This Cross-Assembler has emerged as an aid to assembly language programming on the HP-2100A computer at McMaster. The students have used the Cross-Assembler for debugging their assembly language programs and in some cases for producing the object programs. A typical 100 cards program needs about 2 CPU seconds for processing by the Cross-Assembler.

The author regrets the lack of special features in the Cross-Assembler, such as Macro-processing and Cross-reference directory. These features could not be developed because of lack of time.

The author has mixed feelings about using PASCAL as the programming language for the Cross Assembler. The style of PASCAL makes it a very attractive language for compiler writing. The advanced data structures such as character and alpha variables, record-types and the ability to organize these in arrays and files was a great convenience. Statements like if-then-else, while-do and Repeat-until were found to be very elegant and economical.

Despite these attractive features, the experience
of programming in PASCAL was often painful. The lack of a relocatable compiler made it necessary to re-compile the entire program (including all the procedures) after the slightest change or error, at considerable expense and inconvenience. During the development of the Cross-Assembler a new version of PASCAL was introduced with slight syntax changes which meant incorporating these changes in the source deck. Because PASCAL is currently available only on CDC computers it makes the Cross-Assembler very un-portable.

The author hopes that these practical difficulties of programming in PASCAL will be overcome when the new relocatable compiler is available. It will then be a very attractive language for programming.
APPENDIX A

INSTRUCTIONS FOR USING H.P. CROSS ASSEMBLER

A. Description

The Hewlett-Packard Cross Assembler is a program available on CDC-6400 which can be used to assemble H.P. 2000 series assembly language programs. It checks syntax, produces machine code listing and (optionally) produces a binary deck which can be loaded into the Hewlett-Packard 2100A computer for execution.

B. Control Cards

The following control cards are required to use the program.

```
JOB,CM40000.
ATTACH(PASCAL,ID=............)
ATTACH(HPASMB,ID=............)
PASCAL(LOAD=HPASMB)
DISPOSE(P,*P8)
End of Record

<table>
<thead>
<tr>
<th>HPASMB.</th>
<th>Program Title (Cols 8-68)</th>
<th>Name(Cols 69-80)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASMB,R,L,N,</td>
<td>etc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Program 1)

END

HPASMB

ASMB,A,T,L,B...

(Program 2)

END

END OF FILE
```
C. Control Parameters

The Assembler Control Card (ASMB,...) may contain up to five parameters separated by commas; following 'ASMB' in the first four columns, eg.,

ASMB,R,L,T
ASMB,R,L,N,T,B
ASMB,A,B,L,Z

The parameters, which may appear in any order, have the following significance

A - absolute program
R - relocatable program
L - produce program listing
B - produce punched binary output
T - produce symbol table
Q - print binary records
N - choose N option
Z - choose Z option

For details see HP Assembler Manual, page 4.5

D. Printed and Punched Output

As indicated above, several programs can be assembled in a batch. Each program must have the following structure

HPASMB Program Title(Col. 8-68) Programmer's Name(Col. 69-80)
ASMB,R,L,N, etc.

(PROGRAM DECK)

END

Each program in the batch is assigned a serial number (HP01, HP02,..., etc.) which appears at the top of the program listing and is also punched on a Header Card. This serves to identify the program and its punched output. If an error occurs in a program then its punched output is suppressed but a card reading 'ERROR' is punched out.

+ not available on the standard HP Assembler.
E. Error Messages

If errors are detected in the Assembler Control cards, then the error messages are printed out and the program is not assembled. All the cards up to the next END card ignored. However, the assembly of the remaining programs in the batch (if any) is not affected.

Errors detected in PASSONE are listed on page 1, along with the erroneous statements. A complete program listing is produced during PASSTWO, and any errors found in PASSTWO appear in this listing.
APPENDIX B

ALPHABETIC LIST OF INSTRUCTIONS

ABS Define absolute value
ADA Add to A
ADB Add to B
ALF Rotate A left 4
ALR Shift A left 1, clear sign
ALS Shift A left 1
AND "And" to A
ARS Shift A right 1, sign carry
ASC Generate ASCII characters
ASL Arithmetic long shift left
ASR Arithmetic long shift right
BLF Rotate B left 4
BLR Shift B left 1, clear sign
BLS Shift B left 1
BRS Shift B right 1, carry sign
BSS Reserve block of storage starting at symbol
CCA Clear and complement A (1's)
CCB Clear and complement B (1's)
CCE Clear and complement E (set E = 1)
CLA Clear A
CLB Clear B
CLC Clear I/O control bit
CLE Clear E
ALPHABETIC LIST OF INSTRUCTIONS (cont)

CLF  Clear  I/O  flag
CLO  Clear  overflow  bit
CMA  Complement  A
CMB  Complement  B
CME  Complement  E
CDM  Reserve  block  of  common  storage
CPA  Compare  to  A,  skip  if  unequal
CPB  Compare  to  B,  skip  if  unequal
DEC  Defines  decimal  constants
DEF  Defines  address
DEX  Defines  extended  precision  constants
DIV  Divide
DLD  Double  load
DST  Double  store
ELA  Rotate  E  and  A  left  1
ELB  Rotate  E  and  B  left  1
END  Terminate  program
ENT  Entry  point
ERA  Rotate  E  and  A  right  1
ERB  Rotate  E  and  B  right  1
EQU  Equate  symbol
EXT  External  reference
FAD  Floating  add
FDV  Floating  divide
### ALPHABETIC LIST OF INSTRUCTIONS (cont)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMP</td>
<td>Floating multiply</td>
</tr>
<tr>
<td>FSB</td>
<td>Floating subtract</td>
</tr>
<tr>
<td>HED</td>
<td>Print heading at top of each page</td>
</tr>
<tr>
<td>HLT</td>
<td>Halt</td>
</tr>
<tr>
<td>IFN</td>
<td>When N appears in Control Statement, assemble ensuing instructions</td>
</tr>
<tr>
<td>IEZ</td>
<td>When Z appears in Control Statement, assemble ensuing instructions</td>
</tr>
<tr>
<td>INA</td>
<td>Increment A by 1</td>
</tr>
<tr>
<td>INB</td>
<td>Increment B by 1</td>
</tr>
<tr>
<td>IOR</td>
<td>Inclusive &quot;or&quot; to A</td>
</tr>
<tr>
<td>ISZ</td>
<td>Increment, then skip if zero</td>
</tr>
<tr>
<td>JMP</td>
<td>Jump</td>
</tr>
<tr>
<td>JSB</td>
<td>Jump to subroutine</td>
</tr>
<tr>
<td>LDA</td>
<td>Load into A</td>
</tr>
<tr>
<td>LDB</td>
<td>Load into B</td>
</tr>
<tr>
<td>LIA</td>
<td>Load into A from I/O channel</td>
</tr>
<tr>
<td>LIB</td>
<td>Load into B from I/O channel</td>
</tr>
<tr>
<td>LSL</td>
<td>Logical long shift left</td>
</tr>
<tr>
<td>LSR</td>
<td>Logical long shift right</td>
</tr>
<tr>
<td>LST</td>
<td>Resume list output (follows a UNL)</td>
</tr>
<tr>
<td>MIA</td>
<td>Merge (or) into A from I/O channel</td>
</tr>
<tr>
<td>MIB</td>
<td>Merge (or) into B from I/O channel</td>
</tr>
<tr>
<td>MPY</td>
<td>Multiply</td>
</tr>
<tr>
<td>NAM</td>
<td>Names relocatable program</td>
</tr>
</tbody>
</table>
### ALPHABETIC LIST OF INSTRUCTIONS (cont)

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>No operation</td>
</tr>
<tr>
<td>OCT</td>
<td>Defines octal constant</td>
</tr>
<tr>
<td>ORB</td>
<td>Establish origin in base page</td>
</tr>
<tr>
<td>ORG</td>
<td>Establish program origin</td>
</tr>
<tr>
<td>ORR</td>
<td>Reset program location counter</td>
</tr>
<tr>
<td>OTA</td>
<td>Output from A to I/O channel</td>
</tr>
<tr>
<td>OTB</td>
<td>Output from B to I/O channel</td>
</tr>
<tr>
<td>RAL</td>
<td>Rotate A left 1</td>
</tr>
<tr>
<td>RAR</td>
<td>Rotate A right 1</td>
</tr>
<tr>
<td>RBL</td>
<td>Rotate B left 1</td>
</tr>
<tr>
<td>RBR</td>
<td>Rotate B right 1</td>
</tr>
<tr>
<td>REP</td>
<td>Repeat next statement</td>
</tr>
<tr>
<td>RRL</td>
<td>Rotate A and B left</td>
</tr>
<tr>
<td>RRR</td>
<td>Rotate A and B right</td>
</tr>
<tr>
<td>RSS</td>
<td>Reverse skip sense</td>
</tr>
<tr>
<td>SEZ</td>
<td>Skip if E = 0</td>
</tr>
<tr>
<td>SFC</td>
<td>Skip if I/O flag = 0 (clear)</td>
</tr>
<tr>
<td>SFS</td>
<td>Skip if I/O flag = 1 (set)</td>
</tr>
<tr>
<td>SKP</td>
<td>Skip to top of next page</td>
</tr>
<tr>
<td>SLA</td>
<td>Skip if LSB of A = 0</td>
</tr>
<tr>
<td>SLB</td>
<td>Skip if LSB of A = 0</td>
</tr>
<tr>
<td>SOC</td>
<td>Skip if overflow bit = 0 (clear)</td>
</tr>
<tr>
<td>SOS</td>
<td>Skip if overflow bit = 1 (set)</td>
</tr>
<tr>
<td>SPC</td>
<td>Space n lines</td>
</tr>
</tbody>
</table>
**ALPHABETIC LIST OF INSTRUCTIONS (cont)**

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSA</td>
<td>Skip if sign A = 0</td>
</tr>
<tr>
<td>SSB</td>
<td>Skip if sign B = 0</td>
</tr>
<tr>
<td>STA</td>
<td>Store A</td>
</tr>
<tr>
<td>STB</td>
<td>Store B</td>
</tr>
<tr>
<td>STC</td>
<td>Set I/O control bit</td>
</tr>
<tr>
<td>STF</td>
<td>Set I/O flag</td>
</tr>
<tr>
<td>STO</td>
<td>Set overflow bit</td>
</tr>
<tr>
<td>SUP</td>
<td>Suppress list output of additional code lines</td>
</tr>
<tr>
<td>SWP</td>
<td>Switch the (A) and (B)</td>
</tr>
<tr>
<td>SZA</td>
<td>Skip if A = 0</td>
</tr>
<tr>
<td>SZB</td>
<td>Skip if B = 0</td>
</tr>
<tr>
<td>UNL</td>
<td>Suppress list output</td>
</tr>
<tr>
<td>UNS</td>
<td>Resume list output of additional code lines</td>
</tr>
<tr>
<td>XIF</td>
<td>Terminate an IFN or IFZ group of instructions</td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive &quot;or&quot; to A</td>
</tr>
</tbody>
</table>
APPENDIX C

HP ASSEMBLER INSTRUCTIONS (By Groups)

GROUP 1 MEMORY REFERENCE INSTRUCTIONS

Jump and Increment-Skip

ISZ
JMP
JSB

Add, Load and Store

ADA
ADB
LDA
LDB
STA
STB

Logical

AND
XOR
IOR
CPA
CPB

GROUP 2 REGISTER-REFERENCE INSTRUCTIONS

Shift-Rotate

CLE
ALS
BLS
ARS
BR5
RAL
RBL
RAR
RBR
ALR
BLR
ERA
ERB
ELA
ELB
ALF
BLF
SLA
SLB
No-operation

NOP

Alter-skip

CLA
CLB
CHMA
CMMA
CCA
CGB
CLE
CME
CCE
SEZ
SSA
SSB
INA
INB
SZA
S2B
SLA
SLB
RSS

GROUP 3  INPUT/OUTPUT, OVERFLOW AND HALT INSTRUCTIONS

Input/Output

STC
CLC
LIA
LIB
MIA
MIB
OTA
OTB
STF
GLF
SFC
SFS

Overflow

CLO
STO
SOC
SOS
Halt

HLT

GROUP 4

EXTENDED-ARITHMETIC UNIT (Memory Reference)

MPY  DIV  DLD  DST  FAD  FSB  FMP  FDV

GROUP 5

EXTENDED ARITHMETIC UNIT (Register-Reference)

ASR  ASL  RRR  RRL  LSR  LSL

GROUP 6

PSEUDO INSTRUCTIONS

NAM  ORG  ORR  ORB  END  REP  IFN  IFZ  XIF  COM  ENT  EXT  DEF  ABS  EQU  ASC  DEC  DEX  OCT  BSS  UNL  LST  SKP  SPC  SUP  UNS  HED
APPENDIX D
ERROR MESSAGES

A. Control Card Errors

If any of the following errors are encountered then the assembly is terminated.

1. HPASMB. expected in columns 1-7 of the user's card.
2. ASMB. expected in columns 1-5 of the Assembler control card.
3. Illegal parameter in Assembler Control Card.
4. Control Option Error - Neither A or R or both specified.
5. No output parameter specified.

B. Fatal Errors

The following errors do not terminate assembly, however, binary output (if requested) is suppressed:

1. Illegal character in symbol.
2. Illegal operation.
3. Undeclared variable.
4. Illegal character in operand field.
5. Illegal operation in absolute program.
6. Illegal character in octal constant.
7. Integer constant greater than 177777B.
8. Error in pseudo-op ASC (N<1 or N>28).
9. Non-blank operand field expected.
10. Doubly defined symbol.
11. Too many distinct literals (max=25).
12. Illegal op in multiple instruction.
13. Mixing of A and B type instructions is illegal in a multiple instruction.
15. Illegal sequence of operations in a multiple instruction.
16. Invalid literal.
17. Illegal character in constant.
18. Symbol Table overflow (max=499).
19. An ENT symbol cannot appear in a COM or an EXT statement.
20. An ENT symbol must appear as a Label to an instruction.
21. Too many ENT symbols (max=14).
22. Character following the comma is incompatible with the instruction.
23. Program relocatable operand expected.
25. Sign missing between two terms in operand.
26. Operand is mixture of incompatible relocatable type terms.
27. Absolute operand in a relocatable program exceeds 100B.
28. Operand is negative for an instruction other than ABS.
29. Expression is an L-type literal must be absolute.
30. Literal incompatible with the instruction used.
31. Out of range real constant.
32. Operand address is not on the same page of memory as the instruction (Absolute program).
33. Negative or double relocation in the operand.
C. Non-Fatal Errors

These errors should be treated as warnings; they do not affect assembly.

1. EOF encountered, END assumed.
2. Symbol too long, truncated to five characters.
3. Label illegal in an instruction following REP pseudo.
4. First opcode in an Absolute program does not succeed an ORG statement. Program assembled starting at address 2000 octal.
5. First opcode in a Relocatable program does not succeed a NAM statement.
6. Operand exceeds 1777 octal for a memory reference instruction.
7. Operand exceeds 77 octal for an I/O group instruction.
8. Operand exceeds 17 octal for a register-reference instructions.
10. An IFN or an IFZ follows either an IFN or an IFZ without an intervening XIF. The second pseudo is ignored.
APPENDIX E

ABSOLUTE RECORD FORMAT

NOTE: Pages 76-81 are reproduced from the BCS Manual, Reference 7.
APPENDIX F

RELOCATABLE RECORD FORMAT

<table>
<thead>
<tr>
<th>WORD 1</th>
<th>WORD 2</th>
<th>WORD 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORD LENGTH</td>
<td>ID</td>
<td>CHECKSUM</td>
</tr>
</tbody>
</table>

**Explanation**
- RECORD LENGTH = 9 WORDS
- IDENT = 001
- CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS IN RECORD EXCLUDING WORDS 1 AND 3.

<table>
<thead>
<tr>
<th>WORD 4</th>
<th>WORD 5</th>
<th>WORD 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM</td>
<td>NAME OF PROGRAM</td>
<td></td>
</tr>
</tbody>
</table>

**Explanation**
- SYMBOL: FIVE CHARACTER NAME OF PROGRAM

<table>
<thead>
<tr>
<th>WORD 7</th>
<th>WORD 8</th>
<th>WORD 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENGTH OF MAIN PROGRAM SEGMENT (OR ZERO)</td>
<td>LENGTH OF BASE PAGE SEGMENT (OR ZERO)</td>
<td>LENGTH OF COMMON SEGMENT (OR ZERO)</td>
</tr>
</tbody>
</table>

**Explanation**
- A/C: BINARY TAPE PROCESSOR
- 0 IF ASSEMBLER PRODUCED
- 1 IF COMPILER PRODUCED
RELOCATABLE RECORD FORMAT (continued)

ENT RECORD

<table>
<thead>
<tr>
<th>CONTENT</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 87 015312 4 0 15</td>
<td>RECORD LENGTH = 7-59 WORDS</td>
</tr>
<tr>
<td>15 87 IDENT</td>
<td>IDENT = 010</td>
</tr>
<tr>
<td>15 87 E N T E R I E S</td>
<td>ENTRIES: 1 to 14 ENTRIES PER PROGRAM; EACH ENTRY IS FOUR WORDS LONG.</td>
</tr>
<tr>
<td>15 87 CHECKSUM</td>
<td></td>
</tr>
</tbody>
</table>

WORD 1 | WORD 2 | WORD 3

15 87 015 87 015 87 10 | SYMBOL: 5 CHARACTER ENTRY POINT SYMBOL |
| S Y M B L | R: RELOCATION INDICATOR |
| 0 | 0 IF PROGRAM RELOCATABLE |
| 1 | 1 IF BASE PAGE RELOCATABLE |

WORD 4 | WORD 5 | WORD 6 | R

15 015 87 015 87 0 | WORDS 4 THROUGH 7 ARE REPEATED FOR EACH ENTRY POINT SYMBOL |
| RELOCATABLE ADDRESS FOR SYMBOL | |

WORD 7 | WORD 8 | WORD 9

15 87 10 015 | |
| L | |

WORD 10 | WORD 59

RELOCATABLE ADDRESS
RELOCATABLE RECORD FORMAT (continued)

<table>
<thead>
<tr>
<th>CONTENT</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTR RECORD</td>
<td>RECORD LENGTH = 6-60 WORDS</td>
</tr>
<tr>
<td></td>
<td>IDENT = 100</td>
</tr>
<tr>
<td></td>
<td>ENTRIES: 1 TO 19 PER RECORD; EACH ENTRY IS THREE WORDS LONG</td>
</tr>
<tr>
<td>WORD 1</td>
<td>WORD 2</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>13</td>
<td>0151212</td>
</tr>
<tr>
<td>RECORD LENGTH</td>
<td>IDENT</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>WORD 4</td>
<td>WORD 5</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>15</td>
<td>015</td>
</tr>
<tr>
<td>SYMBOL ID. NO.: NUMBER ASSIGNED TO SYMBOL FOR USE IN LOCATING REFERENCE IN BODY OF PROGRAM</td>
<td></td>
</tr>
<tr>
<td>WORD 7</td>
<td>WORD 60</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>15</td>
<td>015</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>SYM</td>
</tr>
<tr>
<td>WORD 4 THROUGH 6 REPEATED FOR EACH EXTERNAL SYMBOL (MAXIMUM OF 19 PER RECORD)</td>
<td></td>
</tr>
</tbody>
</table>
RELOCATABLE RECORD FORMAT (continued)

**DBL RECORD**

<table>
<thead>
<tr>
<th>CONTENT</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>RECORD LENGTH = 5-60 WORDS</td>
</tr>
<tr>
<td>87</td>
<td>IDENT = 011</td>
</tr>
<tr>
<td>01312</td>
<td>Z/C: BASE/CURRENT PAGE LOADING</td>
</tr>
<tr>
<td>763</td>
<td>= 0 FOR BASE PAGE</td>
</tr>
<tr>
<td>015</td>
<td>= 1 FOR CURRENT PAGE</td>
</tr>
<tr>
<td></td>
<td>NO. OF INST. WORDS</td>
</tr>
<tr>
<td></td>
<td>1 TO 45 LOADABLE INSTRUCTION WORDS PER RECORD</td>
</tr>
</tbody>
</table>

**RELOCATABLE LOAD ADDRESS:**

STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW.

R: RELOCATION INDICATORS:
- 000 = ABSOLUTE
- 001 = 15-BIT PROGRAM RELOCATABLE
- 010 = 15-BIT BASE PAGE RELOCATABLE
- 011 = 15-BIT COMMON RELOCATABLE
- 100 = EXTERNAL REFERENCE
- 101 = MEMORY REFERENCE

**INSTRUCTION WORDS**

- R = 001: 15-BIT PROGRAM RELOCATABLE
- R = 010: 15-BIT BASE PAGE RELOCATABLE
- R = 011: 15-BIT COMMON RELOCATABLE

D/I: INDIRECT ADDRESSING
- 0 = DIRECT
- 1 = INDIRECT

Z/C: BASE/CURRENT PAGE LOCATION OF OPERAND ADDRESS AS DETERMINED BY LOADER.
- 0 = BASE PAGE
- 1 = CURRENT PAGE
RELOCATABLE RECORD FORMAT (continued)

Exploration:
- Record length = 4 words
- IDENT = 101

Explanation:
- S: Relocation indicator for transfer address
  - 0 if program relocatable
  - 1 if base page relocatable

Explanation:
- T: Transfer address indicator
  - 0 if no transfer address in record
  - 1 if transfer address present
### APPENDIX G

**PROGRAM LISTING OF THE CROSS-ASSEMBLER**

(THE FOLLOWING IS A DIRECTORY OF PROCEDURES IN THE LISTING)

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN PROGRAM</td>
<td>127</td>
</tr>
<tr>
<td>ABSTRNCORDS</td>
<td>115</td>
</tr>
<tr>
<td>ADMAIN</td>
<td>104</td>
</tr>
<tr>
<td>BLANKLINES</td>
<td>121</td>
</tr>
<tr>
<td>BLANKPOMMA</td>
<td>92</td>
</tr>
<tr>
<td>BLENDMAPP</td>
<td>47</td>
</tr>
<tr>
<td>CENTHE</td>
<td>121</td>
</tr>
<tr>
<td>CHECKLOC</td>
<td>104</td>
</tr>
<tr>
<td>CHECKIN</td>
<td>45</td>
</tr>
<tr>
<td>CHECKSYS</td>
<td>117</td>
</tr>
<tr>
<td>CONVERT</td>
<td>87</td>
</tr>
<tr>
<td>COPYARRAY</td>
<td>88</td>
</tr>
<tr>
<td>COPYCAR</td>
<td>95</td>
</tr>
<tr>
<td>COUNTDIST</td>
<td>108</td>
</tr>
<tr>
<td>CURCHAP</td>
<td>101</td>
</tr>
<tr>
<td>DATECRODS</td>
<td>119</td>
</tr>
<tr>
<td>DIGITZ</td>
<td>42</td>
</tr>
<tr>
<td>ENDCHAP</td>
<td>42</td>
</tr>
<tr>
<td>ENDCRODS</td>
<td>120</td>
</tr>
<tr>
<td>ENTER</td>
<td>69</td>
</tr>
<tr>
<td>ENTRNAME</td>
<td>121</td>
</tr>
<tr>
<td>ENTRCOND</td>
<td>118</td>
</tr>
<tr>
<td>ERRCKECK</td>
<td>106</td>
</tr>
<tr>
<td>ERDFIT</td>
<td>106</td>
</tr>
<tr>
<td>ERRMESSA</td>
<td>89</td>
</tr>
<tr>
<td>ERPRT</td>
<td>89</td>
</tr>
<tr>
<td>EXPAND</td>
<td>93</td>
</tr>
<tr>
<td>EXTRCOND</td>
<td>119</td>
</tr>
<tr>
<td>FLITICAL</td>
<td>152</td>
</tr>
<tr>
<td>GETASC</td>
<td>107</td>
</tr>
<tr>
<td>GETCHIE</td>
<td>107</td>
</tr>
<tr>
<td>GETCOUST</td>
<td>46</td>
</tr>
<tr>
<td>GFVLAFL</td>
<td>101</td>
</tr>
<tr>
<td>GETOUP</td>
<td>99</td>
</tr>
<tr>
<td>GETNAME</td>
<td>99</td>
</tr>
<tr>
<td>GETOT</td>
<td>46</td>
</tr>
<tr>
<td>GETOUPG</td>
<td>101</td>
</tr>
<tr>
<td>GETXIF</td>
<td>108</td>
</tr>
<tr>
<td>GROUND</td>
<td>106</td>
</tr>
<tr>
<td>INCHECQUT</td>
<td>106</td>
</tr>
<tr>
<td>INCHENDL00</td>
<td>108</td>
</tr>
<tr>
<td>KLEAS</td>
<td>106</td>
</tr>
<tr>
<td>LIE</td>
<td>106</td>
</tr>
<tr>
<td>LITHAL</td>
<td>103</td>
</tr>
<tr>
<td>LICTYP</td>
<td>93</td>
</tr>
</tbody>
</table>

**MAINPROGAM**

**MULTICUL**

**MULTIST**

**MULTPRINT**

**NAMECROD**

**NEWPAGE**

**NEXTCHAC**

**NEXTCHET**

**OUTCH**

**OUTPNT**

**OUTCRT**

**PASONE**

**RSTWO**

**POWCH**

**PREF**

**PRESFOUT**

**PRINTARY**

**PRINTERD**

**PRINTTLE**

**RSTTITLE**

**PUNCHCARD**

**PUNCHRECORD**

**PUSH**

**READARRAY**

**READLINE**

**REPACK**

**REPINST**

**SCANOPABN**

**SCANOPR**

**SEARCH**

**SKIPLANKS**

**SKIPLINES**

**SKELEST**

**SKEPIBLANK**

**SKEPTOD**

**SPLACE**

**SQUEEZE**

**STORE**

**STOREINT**

**STOREL**

**STOREPOL**

**STOREPOL**

**STORENO**

**TITLE**

**WHITEL**
************* H.P. CROSS-ASSEMBLER, VERSION C, AUGUST 1974 *************

AUTHOR K. AHMED, DEPT. APPLIED MATHS., McMaster Univ., Hamilton, Ont.

PASCAL COMPILE OPTIONS

- SA-, D-, I-, R-, X-, C-

<table>
<thead>
<tr>
<th>CONS</th>
<th>NO. OF LINES PER OUTPUT PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LIT</th>
<th>SIZE OF LITERAL TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIT</td>
<td>1593</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYM</th>
<th>MAX. NO. OF ENTRY SYMBOLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERR</th>
<th>MAX. NO. OF ERRORS LISTED PER LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EXT</th>
<th>MAX. NO. OF EXTERNAL SYMBOLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIG</th>
<th>NO. OF DIGITS TO BE PRINTED BY OUTOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIG</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INIT</th>
<th>NO. OF INSTRUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FILEOUT</th>
<th>FOR INITIALIZING THE LITERAL TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILEOUT</td>
<td>E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TYPE</th>
<th>BIT = 0.1:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ARRAY(1..3) OF INTEGER:</td>
</tr>
<tr>
<td></td>
<td>ARRAY(1..16) OF INTEGER:</td>
</tr>
<tr>
<td></td>
<td>ARRAY(1..9) OF ALFA:</td>
</tr>
<tr>
<td></td>
<td>ARRAY(1..2) OF CHAR:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INDEX</th>
<th>1..7</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PRECISION</th>
<th>(SINGLE, DOUBLET)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>= RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LN, LOCFLD, OPFLD : INTEGER;</td>
</tr>
<tr>
<td></td>
<td>END!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OUTCODE</th>
<th>= RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LOC, WORD, REF: INTEGER; TYPE: CHAR;</td>
</tr>
</tbody>
</table>

FILEOUT = FILE OF OUTCODE;

<table>
<thead>
<tr>
<th>VAR</th>
<th>EA, PC, LI, SH, NT, PP, ER, NER, NPROC, NUM : INTEGER;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WORD, CODE, DC, CN, LB, CM, CT, CNP, NR, KDH, KL, COL, X, Y : INTEGER;</td>
</tr>
<tr>
<td></td>
<td>TYP, I, ORT, CHAR:</td>
</tr>
<tr>
<td></td>
<td>LB, PROP, ALFA:</td>
</tr>
<tr>
<td></td>
<td>MAX, MIN, E: REAL:</td>
</tr>
<tr>
<td></td>
<td>PFA, ENTRINPUT, PRINTERR, LISTOUTPUT : BOOLEAN;</td>
</tr>
<tr>
<td></td>
<td>ERFLAG, INPUT, REPL, DUP, ECHN, TPRINT : BOOLEAN;</td>
</tr>
<tr>
<td></td>
<td>SCRM, SCRAM, LISTFIN, GNOUTPUT, FATAL : BOOLEAN;</td>
</tr>
<tr>
<td></td>
<td>ERSPRO, PRINT, LISTFLAG, SUPFL, PAGEFL, EXTRALINE : BOOLEAN;</td>
</tr>
</tbody>
</table>

| CSET | SET OF 15, 36: |
|      | PSEUDOSUBSET: SET OF 23: |
|      | PARAMETER, LISTCHARS, ALPHABET, DIGITS: SET OF CHAR; |
|      | NONFATAL, ERPPAGE: SET OF 0..58: |

<table>
<thead>
<tr>
<th>P.R</th>
<th>FILEOUT:</th>
</tr>
</thead>
<tbody>
<tr>
<td>S 1</td>
<td>FILE 12 OF STATEMENT:</td>
</tr>
<tr>
<td>OUT</td>
<td>FILE 12 OF INTEGER:</td>
</tr>
</tbody>
</table>
TABLE OF INSTRUCTIONS (ALPHABETICAL)

TABLE THAT ARRANGES INSTRUCTIONS BY GROUPS

*/
TABLES USED IN SOME OF THE PROCEDURES

In the following sections, tables are used to represent the data in a structured manner. The tables consist of rows and columns, with each column heading indicating the type of data it contains. The rows exhibit the values corresponding to the column headings.

These tables are essential for organizing and analyzing the data efficiently. They provide a clear and concise overview of the information, making it easier to understand the relationships and patterns involved.

At the end of each table, a brief explanation is provided, summarizing the contents and highlighting the key points. This approach helps in quickly grasping the essential information without getting lost in the details.
\[ KT = (1, 2, 3, 1, 2, 1, 0, 2); \]
\[ PT = (3, 2*5, 3*1, 4, 6, 7, 2, 2*3, 8, 3*1, 4, 6, 7); \]
\[ COMP = (1, 2, 3, 4, 5, 6, 16, 11, 14, 74, 75, 76, 78, 79, 80, 81); \]
\[ FLRL = (E, FADE, E, FSRE, E, FMHE, E, FOVE); \]

**Table of ASCII Code for Characters Arranged with Ascending Display Codes**

ASCII Code:

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
</tr>
</tbody>
</table>

**The Following Tables Contain Coded Information for Title Page, Header Card, Error Card, etc.**

**MH = \{ 4*EMH, MHE, 2*EMMHHMMHHHE, 6*EMM, MHE \};**

**PP = \{ 5MMHHHHHH, ENMHHMMHHHE, 2*EMM, MHE, EMMHMMHHE, EMMHMMHHE, 6*EMMHE \};**

**DIG = \{ 2* E M M M M H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H H
PROCEDURE OUTH(CH:CHAR):  
PRINT THE CHARACTER CH +
BEGIN
  OUTPUT+1:CH:PUT(OUTPUT)
END *OUTH*;

PROCEDURE OUTINT(I:INTEGER);
PRINT THE DECIMAL INTEGER I AND A BLANK*
VAR I,J : INTEGER; A[1..20] OF INTEGER;
BEGIN
  I := 0; J := N;
  REPEAT I := I + 1; A[I] := J MOD 10; J := J DIV 10 UNTIL J = 0;
  FOR J := I DOWNTO 1 DO OUTH(CHR(A[I]+27)); OUTH('E');
END *OUTINT*;

PROCEDURE OUTFNT(I:INTEGER);
PRINT N IN OCTAL FORMAT, IF N IS LESS THAN SIX DIGITS THEN LEFT-JUSTIFY*
WITH ZEROES TO MAKE SIX DIGITS*
VAR F : BOOLEAN;
BEGIN
  F := FALSE;
  M := 7; S := 3;
  FOR J := 1 TO 20 DO
    BEGIN
      IF (F) THEN FOR J := 1 TO NO DO WRITE('0') ELSE WRITE('1')
    END;
END *OUTFNT*;

PROCEDURE NEWPAGE;
WRITE ON A NEW PAGE*
BEGIN
  PG := PG + 1; WRITE(EOL,E1,E,EPAGE,E,PG,5,E E);
  LI := 2;
END *NEWPAGE*;
PROCEDURE READARRAY(VAR A: ARRAY CHAR90);  
READ NEXT CARD IMAGE (UPTO THE EOL CHARACTER); STORE IT IN AN ARRAY OF CHARACTERS A; 
VAR I: INTEGER; X: CHAR; 
BEGIN 
  IF EOF(INPUT) THEN ENDINPUT = TRUE 
  ELSE BEGIN 
    I := 0; REPEAT I := I + 1; READ(A(I)) UNTIL (A(I) = EOL) OR (I = 88); 
    IF I = 88 THEN BEGIN A[89] := EOL; REPEAT READ(X) UNTIL X = EOL END 
  END; 
END = READARRAY;  

PROCEDURE PRINTARRAY(A: ARRAY CHAR90);  
PRINT ARRAY OF CHARACTERS A, UPTO EOL;  
VAR I: INTEGER; 
BEGIN 
  I := 0; REPEAT I := I + 1; OUTCH(A(I)) UNTIL A(I) = EOL; OUTCH(E E); 
  L1 := L1 + 1; IF L1 = LP THEN BEGIN 
    NEWPAGE; 
    PRINTARRAY(HDNG); WRITE(EOL, EOF, EOL, E E) 
  END; 
END = PRINTARRAY;  

PROCEDURE COPYARRAY(A: ARRAY CHAR90; VAR A: ARRAY CHAR90);  
COPY THE ARRAY OF CHARACTERS A ONTO ARRAY B;  
VAR I: INTEGER; 
BEGIN 
  I := 0; REPEAT I := I + 1; B(I) := A(I) UNTIL A(I) = EOL 
END = COPYARRAY;  

PROCEDURE SPACE(N: INTEGER);  
PRINT N BLANKS;  
VAR I: INTEGER; 
BEGIN 
  FOR I := 1 TO N DO OUTCH(E E); 
END = SPACE;  

PROCEDURE MULTIPRINT(CH: CHAR; N: INTEGER);  
PRINT THE CHARACTER CH REPEATELY N TIMES;  
VAR I: INTEGER; 
BEGIN 
  FOR I := 1 TO N DO OUTCH(CH) 
END = MULTIPRINT;
1 WRITE('SYMBOL TOO LONG, TRUNCATED TO 3 CHARACTERS!');
2 WRITE('ILLEGAL CHARACTER IN SYMBOL');
3 WRITE('UNDECLARED VARIABLE!');
4 WRITE('ILLEGAL OPERAND! SHORT OPERAND FIELD EXPECTED!');
5 WRITE('ILLEGAL OPERAND IN OPERAND FIELD!');
6 WRITE('ILLEGAL OPERATOR IN AN ABSOLUTE PROGRAM!');
7 WRITE('ILLEGAL CHARACTER IN AN OCTAL CONSTANT!');
8 WRITE('CONSTANT GREATER THAN 17777777E!');
9 WRITE('ERROR IN PSEUDO-OP ASC: N<1 OR N>28E!');
10 WRITE('BLANK OPERAND FIELD EXPECTED!');
11 WRITE('DUPLICATE DEFINED SYMBOLE!');
12 WRITE('TOO MANY rapidly DISTINCT LITERALS: MAX=5, MLT=13!');
13 WRITE('MIXING OF A, B TYPE INSTRUCTIONS IS ILLEGALE!');
14 WRITE('ILLEGAL MIXING OF SRC AND ASG INSTRUCTION!');
15 WRITE('ILLEGAL SEQUENCE OF OPS IN A MULTIPLE INSTRUCTION!');
16 WRITE('INVLID LITERAL!');
17 WRITE('ASPEXPECTED IN COLS. 1 - 5 OF THE CONTROL CARD!');
18 WRITE('CONTROL CARD ERROR: NEITHER A OR R OR BOTH SPECIFIED!');
19 WRITE('CONTROL CARD ERROR: NO OUTPUT PARAMETER SPECIFIED!');
20 WRITE('SYMBOL TABLE OVERFLOW: MAX=6775E!');
21 WRITE('MOVE EXT SYMNO APPEARS IN A COM OR AN EXT STATEMENT!');
22 WRITE('MOVE EXT SYMNO IS NOT A LABEL OF ANY INSTRUCTION!');
23 WRITE('TOO MANY EXT SYMNO: MAX=5, MLT=13!');
24 WRITE('LABEL ILLEGAL IN AN INSTRUCTION FOLLOWING REP PSEUDO!');
25 WRITE('CHARACTER FOLLOWING COMA IS INCOMPATIBLE WITH !');
26 WRITE('THE INSTRUCTION USED!');
27 WRITE('PROGRAM RELOCATABLE OPERAND EXPECTED!');
28 WRITE('ABSOLUTE OPERAND EXPECTED!');
29 WRITE('MISSING BETWEEN TWO TERMS IN OPERAND!');
30 WRITE('EXTERNAL AND COMMON RELOCATABLE TERMS! END!');
31 WRITE('ONLY WRITE=EXTERNAL AND COMMON RELOCATABLE TERMS! END!');
32 WRITE('ABSOLUTE OPERAND IN A RELOCATABLE PROGRAM EXCEEDS 16777216!');
33 WRITE('RELOCATABLE OPERAND IS NEGATIVE FOR AN INSTRUCTION OTHER THAN ABS!');
34 WRITE('EXPRESSION IN AN L-TYPE LITERAL MUST BE ABSOLUTE!');
35 WRITE('LITERAL INCOMPATIBLE WITH THE INSTRUCTION USED!');
36 WRITE('FIRST OPCODE IN AN ABS PROGRAM DOES NOT SUCCEED AN ORG !');
37 WRITE('FIRST OPCODE IN A RELOCATABLE PROGRAM DOES NOT SUCCEED !');
38 WRITE('ONLY WRITE=NAME STATEMENT!');
39 WRITE('PROGRAM ASSEMBLED STARTING AT ADDRESS 2JG3E!');
40 WRITE('OPERADEXCEEDS 1777779 FOR A MEMORY REF. INSTRUCTION!');
41 WRITE('OPERADEXCEEDS 7777 FOR AN I/O GROUP INSTRUCTION!');
42 WRITE('OPERADEXCEEDS 7777 FOR A REGISTER REF. INSTRUCTION!');
43 WRITE('OPERADEXCEEDS 7777 FOR A REGISTER REF. INSTRUCTION!');
44 WRITE('SOUT OF REGISTER CONSTANTS!');
45 WRITE('BEGIN WRITE=AN IFD OR AN IF7 follows either an IFX OR AN IF7!');
46 WRITE('WRITE=WITHOUT AN INTERVENING X: THE SECOND PSEUDO!');
47 WRITE('WRITE=INSTRUCTION IS IGNOREDE!');
48 WRITE('EXPECTED IN COLS. 1 - 7 OF THE USER#S CARD!');
PROCEDURE PRINTERP;

PRINT ERRORS IN THE LAST STATEMENT;

BEGIN
  FOR [i=1 TO NER] DO ERREMESSAGE(ERRSTORE[i]); LINES(1);
END = PRINTERP;

PROCEDURE CURRCHAR(VAR CH:CHAR);

PUT THE CURRENT CHARACTER IN CH. COL IS THE POINTER TO THE CURRENT
CHARACTER. IF SCAN1=TRUE THEN PASS ONE ELSE PASS TWO.

BEGIN
  IF COL>72 THEN CH=EOL ELSE
    CH=INTEXT(COL)
END = CURRCHAR;

PROCEDURE NEXTCHAR(VAR CH:CHAR);

PUT THE NEXT CHARACTER IN CH. (ALSO UPDATE COL).

BEGIN
  COL=COL+11 IF COL>72 THEN CH=EOL ELSE CH=INTEXT(COL)
END = OF NEXTCHAR;

PROCEDURE SKIPBLANKS;

SKIP TO THE FIRST NON-BLANK CHARACTERS.

BEGIN
  CURRCHAR(CH);
  WHILE CH=' ' DO NEXTCHAR(CH)
END = OF SKIPBLANKS;

PROCEDURE SKIPTOBLANK;

SKIP TO THE FIRST BLANK CHARACTER.

BEGIN
  CURRCHAR(CH);
  WHILE CH=' ' DO NEXTCHAR(CH)
END OF SKIPTOBLANK;

PROCEDURE INCRCOUNT(N:INTEGER);

PROCEDURE INCRCOUNT(N:INTEGER);

FUNCTION DIGIT(CH:CHAR):BOOLEAN;
  IS CH A DIGIT?
BEGIN
  DIGIT:=CH IN DIGITS
END OF DIGIT;

FUNCTION BLANKORCOMMA(CH:CHAR):BOOLEAN;
  IS CH A BLANK OR A COMMA?
BEGIN
  BLANKORCOMMA:=CH=E OR (CH=E, E)
END OF BLANKORCOMMA;

FUNCTION BLANKOREOL(CH:CHAR):BOOLEAN;
  IS CH A BLANK OR AN EOL CHAR?
BEGIN
  BLANKOREOL:=CH=E OR CH=EOL
END BLANKOREOL;

FUNCTION ENDOCHAR(CH:CHAR):BOOLEAN;
  IS CH A BLANK OR COMMA OR EOL?
BEGIN
  ENDOCHAR:=CH=E OR (CH=E, E) OR CH=EOL
END ENDOCHAR;

PROCEDURE SKIPREST;
  SKIP ALL THE CHARACTERS TILL A BLANK OR COMMA
BEGIN
PROCEDURE LOCTYP(VAR LOC: INTEGER; VAR TYPE: CHAR);

PUT THE VALUE OF THE CURRENT LOC. COUNTER IN LOC AND THE PAGE INFORMATION IN TYPE. (TYPE=E FOR CURRENT PAGE, TYPE=B FOR BASE PAGE).

BEGIN
  IF PAGEFLAG THEN BEGIN LOC:=LC; TYPE:=E; END;
  IF SCAN1 THEN BEGIN LOC:=RPC; TYPE:=B; END;
END; LOCTYP;

FUNCTION POWER(X: REAL; N: INTEGER) : REAL;

A FUNCTION FOR EXponentiation (which does not exist in PASCAL), POWER=X**N, RESULT IS REAL.

VAR
  W, Z: REAL; N: INTEGER;

BEGIN
  W:=X; Z:=1.0; I:=N;
  WHILE I>0 DO BEGIN
    IF ODD(I) THEN Z:=Z*W;
    I:=I DIV 2;
    W:=SQR(W);
  END;
  IF I>0 THEN POWER:=Z ELSE POWER:=1/W;
END; POWER;

PROCEDURE SQUEEZE(VAR 9: ARRAY CHAR[9]: VAR A: ARRAY ALFA9);


VAR
  I, J: INTEGER;

BEGIN
  J:=0;
  REPEAT J:=J+1 UNTIL B[J]=EOL;
  FOR I:=J TO 9 DO A[I]:=Z[9(J)+EOL];
  J:=1;
  FOR I:=1 TO 9 DO BEGIN PACK(B, J, A[I]); J:=J+1 END
END; SQUEEZE;

PROCEDURE EXPAND(VAR A: ARRAY ALFA9; VAR B: ARRAY CHAR90);

THIS PROCEDURE IS OPPOSITE OF SQUEEZE, I.E. UNPACKS A INTO B.

VAR
  I, J: INTEGER;

BEGIN
  J:=1; FOR I:=1 TO 9 DO BEGIN UNPACK(A[I], B, J); J:=J+10 END;
END; EXPAND.
PROCEDURE READLINE;
* READ THE NEXT SOURCE LINE.*
BEGIN
IF NER=6 THEN ERRORS IN PREVIOUS LINE. PRINT LINE AND ERROR NOS.*
BEGIN
IF FIRSTEPP THEN PRINTTITLE:
WRITE(LINE); PRINTRAY.(INTEXT); PRINTEPR
END;
READARRAY(INTEXT);
IF ENDINPUT THEN BEGIN
ERROR(1); UNPACK(E, END, E, INTEXT, 1);
INTEXT(111)=EOL; SQUEEZE(INTEXT, $+$, LINE)
END
ELSE BEGIN
LN=LN+1; $+$, LN=LN; SQUEEZE(INTEXT, S+$, LINE);
IF INTEXT(111)$=EOL THEN COMMENT CARD, READ ANOTHER LINE.*
BEGIN $+$, OPFD(0)=U; $+$, LOCFD(0)=LC; PUT(S)+ GOTO 1 END;
END BEGIN;
*************
PROCEDURE WRITELINE;
* WRITE THE CURRENT LINE, PRINT ERRORS, IF ANY.*
PROCEDURE WRITEOUT;
VAR I:INTEGER;
BEGIN
FOR I:=1 TO 9 DO WRITE($+$, LINE(I)); OUTCH(E, E);
LINE(I):=LINE(I)+IF LI+LP THEN TITLE
END IFPRINTOUT;*
BEGIN
IF LSTFLAG THEN WRITEOUT
ELSE IF NER#0 THEN BEGIN
LINES(1); WRITE(LN); SPACE(2);
WRITEOUT
END
IF NER#0 THEN PRINTERR
END IFWRITELINE;
*************
PROCEDURE STORELINE;
* STORE INFORMATION ABOUT THE CURRENT STATEMENT ON A FILE OF RECORD $+$,
FOR LATER USE IN PASS-TWO.*
BEGIN
IF PAGEFLAG THEN S+$, LOCFD(0)=LC ELSE S+$, LOCFD(0)=BPC;
S+$, OPFD(0)=OPN
END IFSTORELINE;*
*************
PROCEDURE COPYCARD;

* WRITE LINE WITH PROPER SPACING

BEGIN
  SPACE(20); WITELINE
END *COPYCARD*

PROCEDURE STOREWORD(L, T INTEGER; T: CHAR; MR INTEGER);

* STORE LOC. COUNTER, INSTRUCTION CODE, AND TYP (I.E., RELOCATIBILITY OF * THE OPERAND) ON FILE R (IF CURRENT PAGE), OR FILE B (IF BASE PAGE)

VAR \( k, n \) INTEGER

PROCEDURE PUTF(VAL FILE OUT CODE);

BEGIN
  WITH F DO BEGIN LOC = L; WORD = W; TYP = T; MREF = MR END; PUTF(F) *
END *PUTFILE*

BEGIN
  IF BINOUTPUT THEN
  IF PAGEFLAG THEN PUTF(L) ELSE PUTF(B);

  IF LISTFLAG \& NERIC0 THEN
  IF EXTRALINE THEN BEGIN
    IF SUPFLAG THEN BEGIN
      SPACE(12); OUTOC(T(L));
      OUTOC(T(W)); OUTCH(T); LINES(1)*
      END;
    IF NERIC0 THEN PRINT;
  END ELSE BEGIN
    NERIC = N;
    IF MREF \& MOD 2610B THEN BEGIN
      PRINT.styleable;
      BEGIN K := MOD MOD 2C1DB; APPEND(N, 0, K) END;
      OUTOC(T(L)); OUTOC(T(N)); OUTCH(T); SPACE(5);*
      WITELINE
    END;
  END

END *STOREWORD*

PROCEDURE CHECKSIGN(VAR S: BOOLEAN);

* CHECK SIGN OF A NUMBER AND ADVANCE TO NEXT CHARACTER

BEGIN
  CURRCHAR(CH);
  SI = TRUE;
  IF CH = - THEN NEXTCHAR(CH) ELSE
  IF CH = + THEN BEGIN SI = FALSE; NEXTCHAR(CH) END
END OF CHECKSIGN*;
PROCEDURE GETOCT(VAR Ni:INTEGER);
EVALUATE AN OCTAL INTEGER CONSTANT, CHECK FOR ERRORS.
VAR S:BOOLEAN; CH:CHAR; Ni:INTEGER;
BEGIN
  SKIPBLANKS; CHECKSIGN(S); Ni:=21;
  WHILE ENDCCH.^(CH#7777) DO BEGIN Ni:=Ni*8+ORD(CH)-27; NEXTCHAR(CH) END;
  IF DIGIT(CH) THEN ERROR(7);
  IF Ni GT 1777777 THEN BEGIN Ni:=3; ERROR(1); END;
  IF "S THEN Ni:=200; SB-N :=S COMPLEMENT*;
END OF GETOCT*;

---------------------------------------------

PROCEDURE GETNUM(VAR Ni:INTEGER);
EVALUATE AN INTEGER. TREAT IT AS OCTAL IF FOLLOWED BY B +
VAR S:BOOLEAN; I,J,BASE:INTEGER;
ARRAY(I..5) OF CHAP;
BEGIN
  Fi:=FALSE; I:=O; Ni:=O;
  SKIPBLANKS; CHECKSIGN(S);
  WHILE DIGIT(CH) DO BEGIN
    IF I<5 THEN BEGIN I:=I+1; A[i]:=CH END ELSE F:=TRUE;
    NEXTCHAR(CH) END;
    IF F THEN GOTO 3;
    IF CH#EQ THEN BEGIN BASE:=8; NEXTCHAR(CH) END ELSE BASE:=10;
    FOR J:=1 TO 1 DO Ni:=Ni*BASE*ORD(A[J])-27;
    IF Ni GT 1777777 THEN BEGIN ERROR(8); Ni:=J; GOTO 2 END;
    IF S THEN Ni:=200; SB-N :=S COMPLEMENT*;
  END OF GETNUM*;

---------------------------------------------

PROCEDURE GETCONST(VAR KIND:INTEGER; VAR X:REAL; VAR Ni:INTEGER);
EVALUATE THE NEXT CONSTANT, REAL OR INTEGER. THE RESULT IS CODED AS
FOLLOWS
  KIND=0      ERROR IN CONSTANT, RETURN X=0.0, Ni=O,
  KIND=1      INTEGER CONSTANT N, X=C.0,
  KIND=2      REAL CONSTANT, X=MANTISSA, N=EXponent TO BASE 10
VAR S:BOOLEAN; F,W:REAL;
BEGIN
  KIND:=21; X:=0.9;
  SKIPBLANKS; CHECKSIGN(S);
  IF DIGIT(CH) THEN BEGIN
    GETNUM(Ni); IF S THEN Ni:=N; Xi:=N;
    IF CH#EQ THEN GOTO 3
    ELSE IF CH#EQ THEN BEGIN
      NEXTCHAR(CH);
      IF CH#EQ THEN GOTO 3
      ELSE IF DIGIT(CH) THEN GOTO 4
      ELSE IF ENDCCH THEN GOTO 5
  END OF GETCONST*;

---------------------------------------------
ELSE GOTO 2 END
ELSE IF ENDOCHAR(CH) THEN BEGIN KIND=1; XI=C.0; GOTO 6 END END
ELSE IF CH=E.E THEN BEGIN NEXTCHAR(CH): IF DIGIT(CH) THEN GOTO 4 ELSE GOTO 2 END
ELSE
ELSE IF ILLEGAL CHARACTER THEN
ELSE IF CH=D THEN BEGIN NEXTCHAR(CH): IF DIGIT(CH) THEN BEGIN GETNUM(N); IF S THEN N1=-N1; IF ENDOCHAR(CH) THEN GOTO 6 ELSE GOTO 2 END
END
ELSE IF CH=PERIOD THEN BEGIN IF !=1.0; YI=Y: WHILE DIGIT(CH) DO BEGIN F1=F/10; YI=Y+F*(ORD(CH)-27); NEXTCHAR(CH) END; IF S THEN YI=-YI; XI=X*Y; ELSE IF ENDOCHAR(CH) THEN GOTO 5 ELSE GOTO 2 END
N1=0; IF BLANK OR COMMA THEN BEGIN
IF X=0 THEN BEGIN YI=ABS(X*POWER(10.0,N1)); IF Y<MINREAL OR Y>MAXREAL THEN ERROR(44) END
END
BEGIN
PROCEDURE CONVERT(P:PRECISION; VAR XI:REAL; VAR N,W1,W2,W3:INTEGER); VAR N1:INTEGER; S:BOOLEAN;
BEGIN
IF X<0.0 THEN BEGIN W1=0; W2=0; W3=0; GOTO 3 END ELSE IF X<0.0 THEN BEGIN S=FALSE; XI=-X END ELSE S=TRUE;
X1 = X * POWER(1.25, N); N1 = 3 * N;  # EXPOENT TO 2
# NORMALIZE FRACTION
WHILE X1 .LT. 1 DO BEGIN X1 = X / 1.25; N1 = N + 3 END;
WHILE X1 .LE. 1 DO BEGIN X1 = X / 1.25; N1 = N - 3 END;
WHILE X1 .EQ. 0.5 DO BEGIN X1 = 2 * X; N1 = N - 1 END;
X1 = X * EXPON; IF X1 .GT. 999999 THEN BEGIN X1 = 0.5; N1 = N + 1 END;
# EXPOENT
N1 = 2 * N;
IF N > C THEN N1 = N + 4 (C1);  # 2'S COMPLEMENT IF EXPOENT IS NEGATIVE
IF P = DOUBLE THEN # EXTENDED PRECISION GOTO 2;
# ORDINARY PPRECISION CONSTANT
M1 = TRUNC(X * 40000000000);  # 2'S COMPLEMENT IF NEGATIVE
W11 = M1 DIV 400; W12 = M1 MOD 400; APPEND(W2, 8, N);
# SINGLE PRECISION CONSTANT COMPLETED
GOTO 3;
# EXTENDED PRECISION CONSTANT TO BE STORED IN THREE H.P. WORDS
M1 = TRUNC(X * 10000000000000000000);
IF S THEN M1 = -M1;
# PREPARE FIRST WORD
W21 = M1 DIV 1000; W22 = M1 MOD 1000;
# PREPARE SECOND WORD
W21 = W21 DIV 100; W21 = W21 MOD 100;
# START THIRD WORD
APPEND(W3, 8, N);
# EXTENDED PRECISION CONSTANT COMPLETED
END # CONVERT;
BEGIN
  CONVERT(P, X, N, W1, W2, W3);
  STOREWORD(LC, W1, E = EP); EXTRALINE1 = TRUE;
  STOREWORD(LC+1, W2, E = EP);
  IF P=SINGLE THEN (LC = LC+2)
    ELSE BEGIN STOREWORD(LC+2, W3, E = MR); LC = LC + 3 END;
END OF STOREREAL*;
============================================================================================

PROCEDURE GETNAME(var A:ALPHA);
*GET THE NEXT SYMOL.* MUST START WITH AN ALPHA OR A PERIOD, CAN CONTAIN
UPTO FIVE ALPHANUMERICS OR PERIODS, LONG SYMBOLS ARE TRUNCATED TO
FIVE CHARACTERS AND A MESSAGE PRINTED OUT*
VAR II:INTEGER; FI:BOOLEAN; CI:ARRAYCHARIC;
BEGIN
  SKIPPLANKS: F1 = FALSE;
  IF CH IN ALPHA THEN C(I) = CH
  ELSE BEGIN
    ERROR(2); A1 = EMPTY;
    REPEAT NEXTCHAR(CH) UNTIL (CH#E=S)*(CH#E.=);
    GOTO 1
  END;
  NEXTCHAR(CH); II = 1;
  WHILE (CH IN ALPHA) OR (CH IN DIGITS) DO
  BEGIN
    IF I < 5 THEN BEGIN I = I + 1; C(I) = CH END ELSE F1 = TRUE;
    NEXTCHAR(CH);
    END;
    IF F THEN ERROR11;
    REPEAT II = I + 1; C(II) = E UNTIL I = 10;
    PACK(C, II, A);
  END OF GETNAME *;
============================================================================================

PROCEDURE SKIPTOEND;
*SKIP ALL THE SOURCE STATEMENTS UNTIL THE END PSUEDO.* USED TO SKIP A
PROGRAM WITH ERRORS IN CONTROL STATEMENT*
VAR A:ALPHA;
BEGIN
  REPEAT
    READARRAY(INTEXT);
    IF EOF(OUTPUT) THEN BEGIN ERROR(0); GOTO 1 END;
    COL = 1; SKIPTOBLANK; GETNAME(A)
    UNTIL A = END;
  END *SKIPTOEND*;
============================================================================================

PROCEDURE ENTERIO(A:ALPHA; II:INTEGER; CI:CHAR);
IN THE SYMBOL TABLE USING HASH-CODING.* A CHECK IS MADE FOR DUPLICATE
ENTRIES. THE RELOCATIBILITY OF THE SYMBOL IS CODED AS FOLLOWS:*
P=PROG, RELOCATABLE, B=BASE PAGE RELOCATABLE, C=COMMON, X=EXTERNAL, 
~ BLANK=ABSOLUTE ~

VAR 
F, I BOOLEAN; 
H, O INDEXES;

BEGIN 
F:=FALSE; OI:=1; 
HI:=(ABS(ORD('I')))) DIV 4096) MOD SZ; 
REPEAT IF SYMTAB(I).NAME=ID THEN ~FOUND+ BEGIN ERROR(11); F:=TRUE END 
ELSE IF SYMTAB(I).NAME=EMPTY THEN 
BEGIN *NEW ENTRY* 
F:=TRUE; 
SN:=SN+1; SYMTP[SN]:=H; 
WITH SYMTAB(H) DO 
BEGIN NAME:=ID; LOC:=L; TYP:=T END 
END ELSE 
BEGIN *COLLISION* 
HI:=H+0; OI:=O+1; 
IF O=SZ THEN BEGIN 
F:=TRUE; ERROR(23) 
END; 
IF H>SZ THEN HI:=H MOD SZ; 
END;

UNTIL F; 
END. CO; ENTER +;

==procedure searchid#91a: var loc:integer; var typ:char; ==
CHECK IF THE GIVEN SYMBOL IS PRESENT IN THE SYMBOL TABLE. IF PRESENT 
THEN, EXTRA flirting LOCATION AND RELOCATABILITY AND TURN THE BOOLEAN 
VARIABLE. PRESENT=TRUE, OTHERWISE SIMPLE TURN PRESENT=FALSE.
VAR 
F BOOLEAN; 
H, O INDEXES;

BEGIN 
F:=FALSE; OI:=1; 
HI:=(ABS(ORD('I')))) DIV 4096) MOD SZ; 
REPEAT IF SYMTAB(H).NAME=ID THEN ~FOUND+ BEGIN F:=TRUE; PRESENT:=TRUE; 
LOC:=SYMTAB(H).LOC; 
TYP:=SYMTAB(H).TYP 
END ELSE IF SYMTAB(H).NAME=EMPTY THEN 
BEGIN *ABSENT* 
F:=TRUE; PRESENT:=FALSE; LOC:=0; TYP:=3 
END ELSE 
BEGIN *COLLISION* 
HI:=H+0; OI:=O+1; IF O=SZ THEN BEGIN TABLE QUASI FULL + 
F:=TRUE; PRESENT:=FALSE 
END; 
IF H>SZ THEN HI:=H MOD SZ;
END;

END SEARCH

================================================================================================

PROCEDURE GETLABEL:

EXTRACT LABEL AND STORE IT IN THE ALFA VARIABLE LBL. IF NO LABEL PRESENT THEN SET LBL=EMPTY (I.E. BLANK WORD)

BEGIN
  COL1=1;
  IF INTEXT(11) = THEN LBL=EMPTY
  ELSE BEGIN
    GETNAME(LBL);
    IF CH=# THEN BEGIN ERROR(2); SKIPTOBLEAN END
    END;
    SKIPBLANKS;
    END OF GETLABEL

================================================================================================

PROCEDURE ENTERLABEL:

ENTER LABEL IN THE SYMBOL TABLE ALONG WITH THE CURRENT LOC. COUNTER AND PAGE INFORMATION

BEGIN
  IF LBL=EMPTY THEN BEGIN
    LOCYPE(LOC,TYP);
    ENTER(LBL,LOC,TYP)
  END

END OF ENTERLABEL

================================================================================================

PROCEDURE GETTOP(VAR OPNIINTEGER);

EVALUATE AND DECODE THE OP FIELD. AN OP MAY APPEAR ALONE OR THERE MAY BE SEVERAL OPs; SEPARATED BY COMMAS TO FORM A MULTIPLE INSTRUCTION

VAR

PROCEDURE CHECKOP(VAR N IINTEGER): LOCAL PROCEDURE

VAR I,J,K IINTEGER; OPALFA; C IARRCHAR10;

BEGIN
  EXTRACT OP FIELD
  I=1;
  REPEAT
  I=I+1; C[I]=CH; NEXTCHAR(CH)
  UNTIL ENDCCHAR(CH);
  REPEAT I=I+1; C[I]=E UNTIL I>10;
  PACK(C,1,OP);
  CHECK OP IN OPTABLE
  K=0; J=N;
  REPEAT
    K=(I+J) DIV 2; * BINARY SEARCH
    IF CPTABLE[K] < OP THEN I=K+1;
    IF CPTABLE[K] > OP THEN J=K-1;
  UNTIL OP=CPTABLE[K]
UNTIL I > J:

  IF VALID OP THEN GET ITS SERIAL NUMBER ELSE SERIAL NO = 0
  IF OPTABLE(k) = OP THEN N1 = OPNUM(k) ELSE N1 = 0
END //CHECKOP------------------

PROCEDURE DUMMYLINE(OPN:INTEGER):
BEGIN
  S LIN = LIN; S • LOCFLO = LC; S • OPFLO = -OPN;
  SOCKET (INTEXT, S • LINE);
END //DUMMYLINE---------------------

BEGIN //MAIN PROCEDURE
CHECKOP(OPN):
IF (CH = E, Z) • (OPN < C) THEN MULTIPLE INSTRUCTION
BEGIN
  OPN = -OPN; STORELINE; PUT(S);
  WHILE CH = E, Z DO
  BEGIN
    NEXTCHAR(CH); CHECKOP(OPN);
    IF OPN = C THEN OPN = 1;
    DUMMYLIKE(OPN); PUT(S);
  END:
  DUMMYLINE(); OPN = -OPN;
END;
END //GETOP••

-----------------------------------------------

PROCEDURE GETASC(VAR N:INTEGER);
STORE THE ASCII CODES FOR THE NEXT TWO CHARACTERS IN A SINGLE WORD N.
THE CODE FOR THE FIRST CHAR. GOES IN THE HIGH 8 BITS AND THE CODE FOR
THE SECOND CHAR. GOES IN THE LOW 8 BITS OF THE HP WORD.

VAR C1, C2: CHAR; N: INTEGER;
BEGIN
  NEXTCHAR(C1); NEXTCHAR(C2);
  N = ASCII CODE (C1) • ASCII CODE (C2);
  APPEND (N, 8, 1) • SHIFT N 8 BITS TO LEFT, TAKE LOGICAL OR WITH I
END //GETASC••

-----------------------------------------------

PROCEDURE SCANOPERAND(VAR NUM:INTEGER; VAR TYPE:CHAR);
SCAN THE OPERAND FIELD AND EVALUATE ADDRESS OR CONSTANT IN THE OPERAND.

VAR N, COUNT, NET, SIGN: INTEGER; T:CHAR; FIRST: BOOLEAN;

-----------------------------------------------

PROCEDURE FLITERAL(VAR NUM:INTEGER);
ROUTINE TO EVALUATE AND STORE FLOATING-POINT LITERALS.

VAR I, N, KIND, W1, W2: INTEGER; X: REAL;
BEGIN
NUM=0;
NEXTCHAR(CH);
GETCONST(KIND,X,N);
IF KIND=2 THEN "VALID FLOATING POINT LITERAL"
BEGIN
CONVERT(SINGLE,X,N,M1,M2,I);
FOR II=1 TO M1-1 DO
BEGIN
 IF (LITTAB[I,1]=M1) AND (LITTAB[I+1,1]=M2) THEN "LITERAL PRESENT"
 THEN LITERAL PRESENT END
 ELSE IF LITTAB[I,1]=NOLIT THEN "NEW LITERAL"
 BEGIN
 LITTAB[I,1]=M1; LITTAB[I,2]=ENOLC;
 LITTAB[I+1,1]=M2; LITTAB[I+1,2]=ENOLC+1;
 NUM=ENOLC; ENOLC=ENOLC+2; GOTO 1
 END END;
 IF ERROR(12) THEN "ERROR(12)"
 ELSE IF ERROR(17) THEN "ERROR(17)"
 END

PROCEDURE LITERAL(VAR NUM:INTEGER);
"ROUTINE TO CHECK AND STORE ALL LITERALS"
VAR I,N:INTEGER; T:CHAR;
FUNCTION COMPATIBLE(CH:TCHAR; OPN:INTEGER):BOOLEAN;
"CHECK IF LITERAL IS COMPATIBLE WITH THE INSTRUCTION USED"
VAR I,N:INTEGER; T:BOOLEAN;
BEGIN
 BI=FALSE;
 IF CH=CH THEN BEGIN I=0; N=15 END ELSE BEGIN I=11; N=16 END;
 REPEAT
 I=I+1
 IF OPN=COMP(I) THEN BI=TRUE;
 UNTIL 3 OR (I=N);
 IF COMPATIBLE=BI THEN "COMPATIBLE"
 END;

BEGIN
 NUM=0;
 NEXTCHAR(CH);
 IF NOT (CH IN LITCHAR) THEN ERROR(17)
 ELSE IF COMPATIBLE(CH,OPN) THEN
 BEGIN
 EXTRACT LITERAL;
 CASE CH OF
 THEN GETASC(N);
 BEGIN NEXTCHAR(CH); GETOCT(N) END;
 CASE CH OF
 THEN GETNUM(N) END;
 CASE CH OF
 BEGIN FLITERAL(NUM); GOTO 1 END;
 CASE CH OF
 BEGIN NEXTCHAR(CH); SCANOPERAND(N,T); IF T<=" THEN ERROR(37) "
 END;
 "CHECK OLD LITERALS, STORE IF LITERAL NOT STORED BEFORE"
 FOR I=1 TO M1 DO
IF LITTAQ(I,1)=1 THEN BEGIN VIRTUAL PRESENT 
    NUM:=LITTAQ(I,2); GOTO 1 END 
ELSE IF LITTAB(I,1) = '0' THEN NEW LITERAL 
    BEGIN 
        LITTAB(I,1) := '0'; LITTAB(I,2) := ENDLC; 
        NUM := ENDLC; ENDLC := ENDLC + 1; GOTO 1 
    END ELSE ERROR(38); 
    CH := ENDLC; 
END \LITERAL; 
---------------------------

PROCEDURE AONUM; 
BEGIN 
    IF SIGN = 1 THEN NUM := NUM + N ELSE NUM := NUM - N 
END \AONUM; 
---------------------------

PROCEDURE NEXTTERM(VAR SIGN: N1: INTEGER; VAR T: CHAR); 
EVALUATE THE VALUE (OF ADDRESS) OF THE NEXT TERM IN THE OPERAND 

VAR A: ALFA; 
BEGIN 
    SIGN := 1; T := 1; 
    IF CH = '=' THEN NEXTCHAR(CH) 
    ELSE IF CH = '-' THEN BEGIN SIGN := -1; NEXTCHAR(CH) END 
    ELSE IF "FIRST THEN ERROR(32); 
    IF (CH IN ALPHA) THEN BEGIN \SYMBOL; SEARCH SYMBOL TABLE 
        GETNAME(A); SEARCH(A, N1, 1); 
        IF "PRESENT THEN ERROR(4) 
    END ELSE IF DIGIT(CH) THEN BEGIN 
        GETNUM(N); T := = END 
    ELSE IF CH = '=' THEN BEGIN LOOP(N+1); NEXTCHAR(CH) END 
    ELSE BEGIN ERROR(5); NEXTCHAR(CH) END; 
    FIRST := FALSE; 
END \NEXTTERM; 
---------------------------

\MAIN PROCEDURE BODY -- SCAN OPERAND 
BEGIN 
    ERROR := FALSE; SKIPLANKS; 
    IF CH = END THEN NO OPERAND FOUND BEGIN ERROR(11); GOTO 1 END; 
    NUM := 0; TYP := =; FIRST := TRUE; INDIRECT := FALSE; 
    IF ABS PROP THEN \ABSOLUTE PROGRAM; 
    REPEAT IF EVALUATE EACH TERM OF OPERAND EXPRESSION, KEEP ADDING 
        NEXTTERM(SIGN, N1); AONUM 
    UNTIL ENDCHAR(CH); 
    ELSE \RELOCATABLE PROGRAM 
    BEGIN 
        NET := 0; COUNT := 0; 
    REPEAT 
        NEXTTERM(SIGN, N1+1); COUNT := COUNT + 1; 
        IF T = = THEN AONUM 
        ELSE IF TYP := = THEN BEGIN TYP := T; AONUM; NET := NET + SIGN END 
    ELSE IF TYP = T THEN BEGIN AONUM; NET := NET + SIGN END;
ELSE BEGIN ERROR(33); GOTO 1 END
UNTIL ENOCAR(ch);

*CHECK RANGE*
IF ABSPROG THEN
BEGIN
IF (OPN<14) THEN ERROR(47)
END;
ELSE *RELOCATABLE PROG.*
BEGIN
IF (OPN<14) THEN ERROR(35); 
IF OPN<15 THEN BEGIN IF OPN>12 THEN ERROR(41) END 
ELSE IF OPN>33 BEGIN IF OPN<77 THEN ERROR(42) END
ELSE ERROR; APiTH AND PSEUDO* IF OPN>77777 THEN ERROR(43)
END;

IF CH=E,Z THEN
BEGIN
IF SCANTOCOMMA THEN BEGIN SCANTOCOMMA:=FALSE; GOTO 1 END
ELSE SCANTOCOMMA:=TRUE; GOTO 1 END
ELSE IF OPN<7 THEN ERROR(29)
END;
ELSE BEGIN ERROR(5); ERROR(29) END;
END;
ELSE IF ERRFLAG THEN NUM:=0
END *SCANOPERAND*;

=================================================================================

PROCEDURE MOVECOL();
*USED IN PASS TWO! MOVES THE CURRENT CHARACTER POINTER (COL) TO THE
OPERAND FIELD (SKIPPING OVER THE LABEL AND OP FIELDS)*
BEGIN
COL:=1;
IF INT(1) THEN SKIPTOBLANK;
SKIPBLANKS; SKIPTOBLANK;
END *MOVECOL*;

=================================================================================
PROCEDURE SCANOPR2(VAR N:INTEGER; VAR T:CHAR);
  SCANOPERAND FOR PASS TWO
BEGIN
  MOVECOL: SCANOPERAND(N,T)
END-SCANOPR2;

PROCEDURE MULTINST(VAR WORD:INTEGER);
* EVALUATE THE CODE FOR A MULTIPLE INSTRUCTION. THE INDIVIDUAL OPS WERE
  STORED (ON FILE 5) AS NEGATIVE NUMBERS IN PASS ONE TO DISTINGUISH THEM
  FROM ORDINARY INSTRUCTIONS
LABEL 1;
VAR
  I,S1,K,P,G,KL,PR,GP,NUM,CODE,INTEGER;
  FIRST : BOOLEAN;
  SAVE : ARRAY [1..2] OF INTEGER;
PROCEDURE ERREXIT(N:INTEGER);
BEGIN
  ERROR(): WORD:=0; WHILE S*OPFLO<0 DO GET(S); GOTO EXIT 1
END-ERREXIT*;

PROCEDURE ERRCHECK;
BEGIN
  IF (NUM<39)*(NUM>73) THEN ERREXIT(13)
END-ERRCHECK*;

FUNCTION KCLASS(NUM:INTEGER):INTEGER;
* CHECK TO SEE IF THE OP BELONGS TO GROUP A OR B OR EITHER OF THE TWO:
  GP A = KCLASS=1, GP B = KCLASS=2, EITHER = KCLASS=0
VAR I:INTEGER;
BEGIN
  [1=01 REPEAT I=I+1 UNTIL NUMSJT[I];
  KCLASS=K[I]
END-KCLASS*;

FUNCTION GROUP(NUM:INTEGER):INTEGER;
* CHECK IF OP BELONGS TO SHIFT-ROTATE GROUP OF ALTERNATE-SKIP GROUP OR BOTH:
  SRG - GROUP=1, ASG - GROUP=2, BOTH - GROUP=0
BEGIN
  IF NUMS64 THEN GROUP1:=1
  ELSE IF NUM=57 THEN GROUP=0
  ELSE GROUP1:=2
END-GROUP*;

FUNCTION PREC(NUM:INTEGER):INTEGER;
* CHECK PRECEDENCE OF OP *

BEGIN
  IF NUM<54 THEN BEGIN IF FIRST THEN PRECI=3 ELSE PRECI=6 END
  ELSE PRECI=PT[NUM]
END *PRECI*;

FUNCTION GETCODE(NUM, GP: INTEGER): INTEGER;

* THE SAME OP MAY HAVE DIFFERENT CODES DEPENDING UPON WHERE IT OCCURS IN A MULTIPLE INSTRUCTION. THIS FUNCTION RETURNS THE APPROPRIATE CODE FOR IN OP.*

VAR N, I: INTEGER;
BEGIN
  CASE GP OF
    0: BEGIN N:=C; SI:=SI+1; SAVE(SI):=NUM. END;
    1: IF (*FIRST)* AND (NUM<54) THEN N:=OPCODE2(NUM) ELSE N:=OPCODE(NUM);
    2: IF NUM<57 THEN N:=OPCODE2(NUM) ELSE N:=OPCODE(NUM)
  END
  GETCODE:=N
END GETCODE;

BEGIN *MAIN PROCEDURE MULTINST*;
  WORD1=0; NUM:=OPN; ERRCHECK;
  FIRST:=TRUE; SI:=0; SAVE[1]:=0; SAVE[2]:=0;
  # DETERMINE KCLASS, GROUP AND PREC OF THE FIRST OP IN A MULTIPLE INSTRUCTION
  KLI:=KCLASS(NUM); GP:=GROUP(NUM); PRI:=PREC(NUM);
  # PUT THE CODE OF THE FIRST OP IN WORD*
  CODE1:=GETCODE(NUM, GP): APPEND(WORD, C, CODE); FIRST:=FALSE;
  GET(S); OPNI:=S+. OPFLD;
  WHILE OPN<7 DO
    BEGIN *PROCESS REMAINING OPS, CHECK COMPATIBILITY IN ALL THREE WAYS*;
      NUM:=OPN; ERRCHECK;
      KL:=KCLASS(NUM);
      IF KL=0 THEN KL:=K ELSE IF (KL<>KL) OR (K<>K) THEN ERREXIT(14);
      GI:=GROUP(NUM);
      IF GP=0 THEN GP:=G ELSE IF (GP<>GP) OR (GP<>GP) THEN ERREXIT(15);
      PI:=PREC(NUM);
      IF PR<P THEN PR:=P ELSE ERREXIT(16);
      CODE1:=GETCODE(NUM, GP); APPEND(WORD, D, CODE);
      GET(S); OPNI:=S+. OPFLD;
    END;
    IF SAVE[I]<>0 THEN
      BEGIN
        IF GP=0 THEN GP:=1
        FOR I:=1 TO 2 DO IF SAVE[I]<>0 THEN
          BEGIN CODE1:=GETCODE(SAVE[I], GP); APPEND(WORD, D, CODE) END
        END
      END;
`PROCEDURE PASSONE:
LABEL 2,3:
VAR FLAG, FIRSTED, NZFLAG boolean; SAVELC INTEGER;

PROCEDURE CHECKRELOC:
*CERTAIN PSEUDO INSTRUCTIONS ARE ILLEGAL IN AN ABSOLUTE PROGRAM. THIS 
PROCEDURE IS CALLED EVERY TIME SUCH A PSEUDO IS ENCOUNTERED TO MAKE 
SURE THAT THE PROGRAM IS RELOCATABLE.
BEGIN
IF ARSPROG THEN BEGIN ERROR(6); GOTO EXIT 2 END;
END CHECKRELOC;

PROCEDURE GETXIF:
*IF AN N (OR 7) OPTION IS CHOSEN THEN THIS PROCEDURE IS CALLED TO SKIP 
ALL STATEMENTS BETWEEN AN IFZ (OR IFN) AND XIF PSEUDO.
VAR a11AFLA, F1 boolean;
BEGIN
F1 = FALSE; n1 = FLAG = FALSE;
REPEAT
READLINE; COL = 1; SKIPBLANK; GETNAME(A);
IF (A=IFZ) OR (A=IFN) THEN ERROE(45);
IF A=XIF THEN BEGIN S+; OPFLD=108; F1=TRUE END
ELSE IF A=END THEN BEGIN S+; OPFLD=89; F1=TRUE END
ELSE BEGIN S+; OPFLD=0; PUT(S) END
UNTIL F1;
END GETXIF;

PROCEDURE COUNTCONST(OPN INTEGER):
*THIS PROCEDURE SCANS THE OPERAND FIELD TO DETERMINE THE NATURE OF 
CONSTANT(S) AND INCREMENT LOC COUNTER ACCORDINGLY.
VAR REALCONST: BOOLEAN;
BEGIN
ENTERLABEL; SKIPBLANKS;
REALCONST = FALSE;
REPEAT
NEXTCHAR(CH);
IF (CH=ES) OR (CH=ES) THEN REALCONST = TRUE
UNTIL ENDCCHAR(CH);
CASE OPN OF
*DEC* 86 1 IF REALCONST THEN INCRCOUNT(2) ELSE INCRCOUNT(1);```
OF

IF \text{CH}=E\text{E} \text{THEN GOTO} 1
END COUNTCONST1

PROCEDURE INCREMENTLOC;

ENTER \text{LABEL AND INCREMENT LOC COUNTER BY} \text{1 IF AN ORDINARY INSTRUCTION,} \times \text{INCREMENT LOC COUNTER BY REPNI IF THE INSTRUCTION WAS FOLLOWED BY A REP} \times \text{PSEUDO}.

BEGIN IF REPFLAG THEN BEGIN
REPFLAG=FALSE; INCRCOUNT(REPN);
IF LAL\#EMPTY \text{THEN ERROR}(28)
ELSE BEGIN ENTERLABEL; INCRCOUNT(1) END
END

BEGIN
SCAN1=TRUE; REPFLAG=FALSE; FIRSTWED1=TRUE; SUPFLAG=FALSE;
FLAG=TRUE; PAGEFLAG=TRUE; FIRSTER=TRUE; SCANTOGO=FALSE;
BPC=0; SAVELOC=0; XT1=0; NT1=0; COM=0; ERI=0; N1ZFLA=FALSE;
LINES(3);

1) \text{PROCFSS THE NEXT SOURCE LINE.}

READLINE; GETLABEL;
IF CH=EOL \text{THEN \#BLANK CARD} \text{BEGIN OPNI}=0; STORELINE; GOTO 2 \text{END}
ELSE GETOP(OPNI);

- \text{CHECK TO SEE THAT THE FIRST VALID OPCODE IN AN ABSOLUTE PROGRAM IS}
- \text{FOLLOWED BY AN ORG STATEMENT AND IN A RELOCATABLE PROGRAM BY A NAM}
- \text{STATEMENT.}

IF FLAG \text{THEN IF} OPNI<85 \text{THEN ERROR}(39)
ELSE IF NOT\text{OPNI-85} \text{IN PSEUDOSUBSET} \text{THEN ERROR}(39);

IF OPNI=6 \text{THEN \#ILLEGAL OPERATION} \text{BEGIN ERROR}(31); STORELINE END
ELSE IF OPNI<7 THEN \#MULTIPLE INSTRUCTION \text{INCREMENTLOC}
ELSE BEGIN STORELINE;

IF OPNI=73 \text{THEN \#MACHINE INSTRUCTION \text{INCREMENTLOC}
ELSE IF OPNI=61 \text{THEN \#EXTENDED ARITH. INSTRUCTION}

BEGIN IF REPFLAG THEN
REPFLAG=FALSE; INCRCOUNT(2*REPN);
IF LAL\#EMPTY \text{THEN ERROR}(28)
END IF CPN\#78 \text{THEN \#FAO,FSR,FP,H,FDV}
BEGIN
CHECKRELOC;
A=EFLB(OPNI); SEARCH(A,N,CH);
IF PRESENT THEN BEGIN
XT1=XT1+1; ENTER(A,XT,EX3); EXT(EXT1)=A
END

END
ELSE \* PSEUDO- INSTRUCTION \* CASE OPN OF
\* ARS \* 42 \* BEGIN ENTERLABEL; INCRCOUNT(1) END;
\* ASC \* 83 \* BEGIN ENTERLABEL; SCANTOCOMMA=TRUE; SCANOPERAND(N,TYP);
\* IF \(N>11\) \& \(N<29\) \* THEN INCRCOUNT(N) \* ELSE ERROR(9) \* END;
\* ASS \* 94 \* BEGIN ENTERLABEL; SCANOPERAND(N,TYP); INCRCOUNT(N) END;
\* COM \* 85 \* BEGIN CHECKRELOC;
\* REPEAT NT=0; GETNAME(A);
\* IF CH=3\(\equiv\) THEN BEGIN
\* NEXTCHAR(CH); SKIPBLANKS;
\* GETNUM(N); SKIPBLANKS;
\* IF CH=\(3\equiv\) THEN NEXTCHAR(CH)
\* ELSE SKIPREST
\* END
\* ELSE IF ENDOCHAR(CH) THEN NT=1
\* ELSE SKIPREST;
\* END;
\* ENTER(A,COM,EC); COM=COM+N; COL=COL+1
\* UNTIL BLANKOREOLOC(CH)
\* END;
\* DEC \* 86 \* BEGIN COUNTCONST(OPN);
\* DEF \* 87 \* BEGIN ENTERLABEL; INCRCOUNT(1) END;
\* DEX \* 88 \* BEGIN COUNTCONST(OPN);
\* END \* 89 \* BEGIN
\* ENDLOC=LCL; S+; LNI=-1; NLNI=LNI; PUT(S);
\* SKIPBLANKS; TRANSFR(11)\(\equiv\)C; \(\equiv\)R-BIT\(\equiv\)
\* IF CH=EOL THEN \(\equiv\)NO TRANSFER ADDRESS SPECIFIED\(\equiv\)
\* BEGIN TRANSFR(21)\(\equiv\)T-BIT\(\equiv\); TRANSFR(3)\(\equiv\)C END
\* ELSE BEGIN EVALUATE TRANSFER ADDRESS\(\equiv\)
\* TRANSFR(21)\(\equiv\)1; SCANOPERAND(TRANSFR(3),TYP); \(\equiv\)
\* IF TYP=28 THEN TRANSFR(11)\(\equiv\)2
\* END
\* GOTO 3
\* END;
\* ENT \* 90 \* BEGIN CHECKRELOC;
\* REPEAT NT1=NT+1;
\* IF NT>MAX THEN BEGIN ERROR(27); GOTO 2 END;
\* GETNAME(ENT(NT1),SYM);
\* IF ENDOCHAR(CH) THEN SKIPREST; COL=COL+1;
\* UNTIL BLANKOREOLOC(CH)
\* END;
\* EDU \* 91 \* BEGIN SCANOPERAND(NUM,TYP);
\* ENTER(LBL,NUM,TYP)
\* END;
\* EXT \* 92 \* BEGIN CHECKRELOC;
\* REPEAT GETNAME(A); X1=X1+1;
\* IF X1>MAX THEN BEGIN ERROR(26); GOTO 2 END;
\* ENTER(A,X1,EX); EXT[X11]=A;
\* IF ENDOCHAR(CH) THEN SKIPREST; COL=COL+1
\* UNTIL BLANKOREOLOC(CH)
\* END;
\* MED \* 93 \* S+; LNI=-1;
IFN 94 \begin{verbatim}
BEGIN
  PUT(S); IF NZFLAG THEN ERROR(45) ELSE NZFLAG = TRUE;
  IF OPTION = E THEN GOTO 1 ELSE GETXIF
END;
\end{verbatim}

IFZ 95 \begin{verbatim}
BEGIN
  PUT(S); IF NZFLAG THEN ERROR(45) ELSE NZFLAG = TRUE;
  IF OPTION = E THEN GOTO 1 ELSE GETXIF
END;
\end{verbatim}

LSI 96 \begin{verbatim}
BEGIN
  IF A9SPROG THEN ERROR(39);
  FLAgI = FALSE; CHECKRELOC; SKIPBLANKS;
  IF CH = EOL THEN PROG1 = EMPTY ELSE GETNAME(PROG);
  SKIPBLANKS;
  IF CH = 'E' THEN BEGIN NEXTCHAR(CH); GETNUM(NMN) END *
  ELSE NZFLAG = TRUE;
END;
\end{verbatim}

NAM 97 \begin{verbatim}
BEGIN
  IF A9SPROG THEN ERROR(39);
  FLAgI = FALSE; CHECKRELOC; SKIPBLANKS;
  IF CH = EOL THEN PROG1 = EMPTY ELSE GETNAME(PROG);
  SKIPBLANKS;
  IF CH = 'E' THEN BEGIN NEXTCHAR(CH); GETNUM(NMN) END *
  ELSE NZFLAG = TRUE;
END;
\end{verbatim}

OCT 98 \begin{verbatim}
COUNTCONST(OPN):
\end{verbatim}

ORA 99 \begin{verbatim}
BEGIN CHECKRELOC; PAGEFLAG = FALSE; BASE PAGE; END;
\end{verbatim}

OPG 100 \begin{verbatim}
BEGIN
  IF A9SPROG THEN ERROR(39) ELSE FLAgI = FALSE;
  PAGEFLAG1 = TRUE; CURRENT PAGE;
  SCANOPERAND(NUM, TYP);
  IF A9SPROG(TYP = ;) THEN BEGIN ERROR(30); GOTO 2 END;
  ELSE LC1 = NUM;
  IF SAVELC = 0 THEN SAVELC = LC
END;
\end{verbatim}

ORN 101 \begin{verbatim}
BEGIN
  PAGEFLAG1 = TRUE; CURRENT PAGE;
  SAVELC = 0 THEN BEGIN LC1 = SAVELC; SAVELC = 0 END
END;
\end{verbatim}

REP 102 \begin{verbatim}
BEGIN
  ENTERLABEL;
  REPFLAG1 = TRUE; SCANOPERAND(REPN, TYP);
  IF TYP = THEN ERROR(39);
END;
\end{verbatim}

SKP 103 \begin{verbatim}
  *LN1 = -1
\end{verbatim}

SPC 104 \begin{verbatim}
  *LN1 = -1
\end{verbatim}

SUP 105

UNL 106

UNS 107

XIF 108 \begin{verbatim}
  NZFLAG = FALSE;
\end{verbatim}

SWP 109 \begin{verbatim}
BEGIN ENTERLABEL; INCRCOUNT(1) END;
\end{verbatim}

\end{verbatim}

21 \begin{verbatim}
PUT(S); GOTO 1;
\end{verbatim}

31 \begin{verbatim}
PRINTERR; LINES(3); CHECK FOR ENTRY LABELS AND RELATED ERRORS>
\end{verbatim}

\begin{verbatim}
FOR I = 1 TO NT DO
BEGIN
  SEARCH(ENT(I), SYM, N, TYP);
  IF PRESENT THEN BEGIN
    IF (TYP = CE) OR (TYP = EX) THEN ERROR(24)
    ELSE BEGIN ENT(I).LOC = N; ENT(I).TYP = TYP END
    ELSE ERROR(25)
  END IF NER = 0 THEN BEGIN
    WRITE(ENT(I).SYM); LINES(1); PRINTERR
  END
\end{verbatim}
END;
LINES(4);
WRITE(EEE) ; IF ER=6 THEN WRITE(NO) ELSE WRITE(ER:6)
WRITE(EEE); LINES(3);
END - PASS ONE+;

================================================================================================

PROCEDURE REPINSTR;
*REPEATEDLY STORE SAME CODE N TIMES; USED IN REP PSEUDO (PASS 2) +
VAR I:INTEGER;
BEGIN
FOR I=1 TO REPN-1 DO
BEGIN
  SPACE(12);
  LC:=LC+1; STOREWORD(LC,WORD,TYP,HP)
END;
END - REPINSTR+;

================================================================================================

PROCEDURE PASS2;
BEGIN
TITLE;
ER:=0; SPC:=0; SCAN1:=FALSE; PAGEFLAG:=TRUE; SUPFLAG:=FALSE;
RESET(S);
LNI:=S[LNI]; LC:=S[LC]; LCFLD; OPN:=S[OPN]; OPFLO;
IF IDSTFLAG THEN IF LN=-1 THEN WRITE(EOL,E,E,E) ELSE WRITE(LN,E,E,E,E);
  IF OPN=9 THEN *COMMENT CARD*
  BEGIN SPACE(20); WRITE(LINE); GET(S); GOTO 1 END;
  EXPAND(S++,LINE,INTEXT);
  EXTRALINE:=FALSE; HII:=1;
  IF OPN=0 THEN *MULTIPLE INSTRUCTION*
  BEGIN
    MULTINST(WORD);
    STOREWORD(LC,WORD,E,E,E,HP);
    IF RERFLAG THEN REPINSTR;
    END *MULTIPLE INSTRUCTION*;
ELSE-OPN
IF OPN=3 THEN *MACHINE INSTRUCTION*
BEGIN
CODE:=OPCODE[OPN];
IF OPN=3 THEN *OPERAND EXPECTED*
BEGIN
  SCANOPR2(WORD,TYP); IF INDIRECT THEN CODE:=CODE+10000B;
  IF ABSPROG THEN
  BEGIN
    IF WORD020098 THEN
    BEGIN
      APPEND(CODE,0,20098); WORD:=WORD MOD 20098
    END
  ELSE BEGIN
    RELLOC, PROG+;
    IF (TYP=3E)+(TYP=3E)+(TYP=3E) THEN
    APPEND(CODE,0,20098);
    IF (OPN=14)+(TYP=3E)+(TYP=3E)+(TYP=3E) THEN
  END
  END
  END
END

END - PASS TWO+;

================================================================================================


BEGIN *MEN* REF., HR1=WORD; WORD1=CODE; GOTO 99 END*
END.
APPEND(WORD, G, CODE)
END.
ELSE IF OPCODE THEN BEGIN WORD1=CODE; TYP1=E = END;
991 STOREWORD(LC, WORD, TYP, MP); IF REPFLAG THEN REPINTP;
END -MACHINE INSTRUCTION-.
ELSE IF OPN=1 THEN -EXTENDED ARITH. INSTRUCTION-
BEGIN.
IF OPN=77 THEN *MPY, DIV, DLO, DST* BEGIN
WORD1=OPCODE(OPN1); TYP1=E =
END.
ELSE *FAO, FSB, FMP, FDO* BEGIN
IF ABSPOP THEN ERROR(6);
SEARCH(FLAG[OPN], N, TYP);
WORD1=16C F09; FSB, CURRENT PAGE;
APPEND(WORD, D, N)
END.
STOREWORD(LC, WORD, TYP, MR); EXTRALINE1=TRUE;
SCANOPRZ(N, T); IF INDIRECT THEN N=N+1000009;
STOREWORD(LC+1, N, T, MR); IF REPFLAG THEN BEGIN
REPFLAG=FALSE;
FOR Y1=1 TO REPN-1 DO BEGIN
LC1=LC+2; SPACE(12);
EXTRALINE1=FALSE; STOREWORD(LC, WORD, TYP, MR);
END
END -EXT. ARITH. INSTR.+
ELSE *PSEUDO-OPERATION* BEGIN
CASE OPN OF
ABS 82 1 BEGIN
SCANOPRZ(N, TYP); IF TYP E THEN ERROR(31);
STOREWORD(LC, N, TYP, MR);
END.
ASC 83 1 BEGIN
SCANTOCOMMA=TRUE; SCANOPRZ(NUM, TYP);
IF (NUM+1) NUM+28 THEN BEGIN ERROR(9); COPYCARD END;
ELSE BEGIN
GETASC(N); STOREWORD(LC, N, = E, MR);
LC1=LC+1; EXTRALINE1=TRUE;
FOR Y1=1 TO NUM-1 DO BEGIN
GETASC(N); STOREWORD(LC, N, = E, MR);
LC1=LC+1
END
END
ABS 84 1 IF LSTFLAG THEN BEGIN
OUTOC(T); OUTOC(0);
OUTOC(T); SPACE(6); WRITELINE
END.
CCM+ 85 1 COPYCAR:
DEC+ 86 1 BEGIN
MOVECOL:
REPEAT
GETCONST(KIND,X,N):
CASE KIND OF
0  ERROR(29); STOREINT(N) ENO;
1  STOREINT(N):
2  STOREPEAL(SINGLE,X,N)
END:
CURRCHAR(CH): COL=COL+1
UNTIL BLANKOREOL(CH)
END:
DEF+ 87 1 BEGIN SCANOPR2(N,TYP): IF INDIRECT THEN N=N+1;OOOOG9;
STOREWORD(LC,N,TYP,MR)
END:
DEX+ 88 1 BEGIN
MOVECOL:
REPEAT
GETCONST(KIND,X,N):
CASE KIND OF
0  ERROR(29);
1  BEGIN X1=N; N=0 END;
2  
END:
STOREREAL(DOUBLE,X,N):
CURRCHAR(CH): COL=COL+1
UNTIL BLANKOREOL(CH)
END:
END+ 89 1 GOTO 2:
ENT+ 90 1 COPYCAR:
EQU+ 91 1 IF LSTFLAG THEN
BEGIN
SCANOPR2(N,TYP): OUTOCT(N); SPACE(13); WRITELINE
END:
EXT+ 92 1 COPYCAR:
HEO+ 93 1 BEGIN
II=0; MOVECOL: COL=COL-1; JI=73-COL:
REPEAT
II=II-1; HDNG[I]=INTEXT(COL+1)
UNTIL (HDNG[I]=EOL) OR (I>J)
HDNG[I]=EOL;
TITLE
END:
IFN+ 94 1 COPYCAR:
IF?+ 95 1 COPYCAR:
ALST+ 96 1 IF LSTOUTPUT THEN
BEGIN
LSTFLAG=TRUE; WRITE(EOL,E,LN,E,E,E,E,E,E,E);
COPYCAR
END:
NHM+ 97 1 IF LSTFLAG THEN
BEGIN OUTOCT(0); SPACE(13); WRITELINE END;
OCT+ 98 1 BEGIN
MOVECOL:
REPEAT
GETOCT(N): STOREINT(N):
IF ENDCOL(COL) THEN SKIPREST; COL=COL+1
UNTIL BLANKOREOL(CH)
END:
ORB+ 99 1 BEGIN
PAGEFLAG=FALSE;
PROCEDURE PUNCHCARD(A:ARRAY16); *TRANSFERS THE INFORMATION CODED IN THE ARRAY A ONTO THE PUNCH FILE P. (USED TO PUNCH HEADER AND ERROR CARDS).*

VAR I:INTEGER
BEGIN
FOR I=1 TO 16 DO BEGIN P= A(I); PUT(P) END
END *PUNCHCARD*;

PROCEDURE ABSBINRECORDS;
*PUNCH (AND/OR PRINT) ABSOLUTE BINARY RECORDS.*

VAR C,I,L-LAST;BF;H;CHECKSUM;Z & INTEGER; *GLOBALS -- R,P (FILES), W* *

PROCEDURE PUSH(N:INTEGER);
*INVERT N AND STORE IT IN BF AFTER SHIFTING LEFT 12 BITS. WHEN 5 SUCH
AFTER MADE TRANSFER OF ON PUNCH FILE AND START AGAIN. (INVERSION MEANS
TURNING A CARD COLUMN UPSIDE DOWN)

BEGIN
71=7+1;
APPEND[RF, 12, INVERT(N)];
IF 7=5 THEN BEGIN P+1=RF; PUT(P); RF=7; Z1=3; CI=C+5 END
END - PUSH*

PROCEDURE STORE(N:INTEGER);
VAR Ki:INTEGER;
BEGIN
IF PUNCHIN THEN #BREAK HP WORD AT 6TH BIT AND SEND FOR PUNCHING*
BEGIN
Ki=N DIV 4096; PUSH(K);
K=Ki*N-K*4096; PUSH(K)
END
IF LISTIN THEN #PRINT HP WORD IN OCTAL FORMAT
BEGIN
M1=M+1; OUTOC(N); WRITE(E, E);
IF M=15 THEN BEGIN M=0; LINES(1) END;
END - STORE*

BEGIN
RF=0; Z1=0; M1=0;
RESET(R);

WHILE NOT(EOF(R)) DO
BEGIN #NEW RECORD EACH LOOP*
W[21]=R*t.LCC;
CHECKSUM=R*t.LOC:
INITIALIZE CHECKSUM
LAST=R*t.LCC-1;
C=0;
L=1;
REPEAT
IF R*t.LCC=LAST+1 THEN LAST=LAST+1 ELSE GOTO 1;
L=L+1;
W[1]=R*t.WORD;
STORE INSTRUCTION;
CHECKSUM=CHECKSUM+R*t.WORD;
UPDATE CHECKSUM
GET(R)
UNTIL (L=39)*EOF(R);

L#COMPLETE RECORD AND TRANSFER TO FILE P*;
IF L=2 THEN #RECORD LENGTH
W[1]=0;
INSERTIAN, W[1];
W[1]=CHECKSUM PCE 210000B;
IF LISTIN THEN BEGIN WRITE(=-ABS BINARY RECORD); LINES(2) END;
FOR I=1 TO L+1 DO STORE(W[I]);

#LEFT JUSTIFY LAST WORD*
WHILE Z=0 DO PUSH(0);

#FILL REMAINDER OF THE CARD WITH 7E20ES*
WHILE C=3 DO BEGIN P+1=0; PUT(P); CI=C+5 END
END - PUNCHA958IN*;
PROCEDURE PUNCHRECORD(RECNAME, ALFA, RL: INTEGER);

PUNCH (AND/OR PRINT) ONE RELOCATABLE BINARY RECORD (63 HP WORDS)

VAR I, J, K, C, REM: INTEGER;

GLOBAL -- W

BEGIN
IF LISTBIN THEN • LIST BINARY RECORD
BEGIN
N=0; WRITE(RECNAME); LINES(2);
FOR I=1 TO RL DO
BEGIN
N=N+1; OUTCHR(W[I]); WRITE(' ', ' ');
IF N=8 THEN BEGIN N=0; LINES(1) END;
END; LINES(4)
END;
IF PUNCHBIN THEN • PUNCH BINARY RECORD, RELOCATABLE FORMAT
BEGIN
FOR I=RL+1 TO 60 DO W[I]=0;
C=0;
REPEAT
REM=REM*10+K
FOR I=1 TO 4 DO
BEGIN
N=REM;
FOR J=1 TO C+3 DO APPEND(N, 16, W[J]);
K=K*4;
IF K<16 THEN BEGIN
C=C+4;
J=H[C] DIV D[1];
REM=H[C]-J*D[1];
APPEND(N,15-K, J)
END
ELSE C=C+3;
P+1=N; PUT(P)
END
UNTIL C=60;
END • PUNCHRECORD

FUNCTION CHECKSUM(RL: INTEGER): INTEGER;
• CALCULATE CHECKSUM (ARITH SUM OF ALL WORDS IN THE RECORD EXCLUDING THE
* FIRST AND THE THIRD WORDS)

VAR I, CS: INTEGER;

GLOBAL -- W

CS=W[2];

FOR I=4 TO RL DO CS=CS+W[I];
CHECKSUM=CS MOD 295C008
END • CHECKSUM

PROCEDURE REPACK(AlFA, VAR BIARRAY);
THE NAME A (5 CHARS) IS RESTORED IN THREE WORDS WITH ASCII CODE FOR TWO CHARs IN EACH WORD. ASCII CODE FOR THE 6TH CHAR IS FORCED ZERO.

VAR I,J,N:INTEGER; CHAR:CHAR19;
BEGIN
UNPACK(A,C,1); C(16):=EQL;
FOR I:=1 TO 3 DO
BEGIN
  J:=2*I-1;
  R(I):=ASCII.CODE(C(J));
  N:=ASCII.CODE(C(J+1));
  APPEND(B[I],0,N);
END
END «REPACK»;

==============================================================================

PROCEDURE NAMERCORE;
	FORM A NAM RECORD IN THE ARRAY W[1..6].
VAR P*ARRINT3; #GLOBALS-- W,PROG,ENOLC,BPC,COM.
BEGIN
  W(11):=10'000;
  W(21):=27'000;
  REPACK(PROG,P);
  APPEND(P[3],0,N);
  FOR I:=1 TO 3 DO W[I+3]:=P[I];
  W(71):=ENOLC;
  W(61):=BPC;
  W(91):=COM;
  W(11):=NAM;
  IF NAM=0 THEN W[11]:="" ELSE W[11]:=143;
  FOR I:=12 TO 17 DO W[I]:=L;
  W(31):=CHECKSUM[17];
  PUNCHRECORD:=NAM RECORDS,17)
END «NAMERCORE»;

==============================================================================

PROCEDURE ENTRECORDER;
	FORM ENTRY RECORD.
VAR I,J,K,RL:INTEGER; P*ARRINT3; #GLOBALS-- W,N,T,ENT.
BEGIN
  RL:=4*N+3;
  W(11):=0: INSERT(R,6,W[I]);
  W(21):=10000+N;
  IDENT*ENTRIES
  FOR I:=1 TO NT DO
  BEGIN
    REPACK(ENT[I],SYM,P);
    IF ENT[I].TYPE=''B'' THEN APPEND(P[3],0,1);
    FOR J:=1 TO 3 DO W[K+J]:=P[J];
    W[K+4]:=ENT[I].LOC;
    K:=K+4
  END
END.
PROCEDURE EXRECORD;

FORM EXTERNAL RECORDS, EACH RECORD TO CONTAIN UPTO 19 EXTERNAL SYMBOLS.*

VAR I, J, K, RL, ML, M2: INTEGER; MORE: BOOLEAN; PAINTINT3;

GLOBAL = W, XT, EXT;

BEGIN
M1 := 1;
IF XT > 19 THEN BEGIN MORE := FALSE; M2 := XT END ELSE BEGIN MORE := TRUE; M2 := 19 END;
I := NEW EXRECORD;
PL := 3 * M2 + 1;
W[1] := INSETPE[RL, 8, W[1]];
W[2] := M2 + 1000009;
ENTRIES + IDENT
K := 3;
FOR I := M1 TO M2 DO
BEGIN
REPACK[EXT[I, PL], APPENDP[3, I]];
FOR J := 1 TO 3 DO W[K + J] := P[3];
K := K + 3;
END;
PUNCHRECORD(EXT RECORD, RL);
IF MORE THEN BEGIN
M1 := M2 + 1; M2 := M2 + 20;
IF XT > 1 THEN BEGIN M2 := XT; MORE := FALSE END;
GOTO 1
END;
END "EXRECORD+1

==========================================

PROCEDURE DRLRECORDS(VAR F:FILEOUTCODE; PG9BIT);

THE DRL RECORDS CONTAIN THE ASSEMBLED CODE. TWO TYPES OF RECORDS MAY BE PRODUCED, VIZ. BASE-PAGE RECORDS AND CURRENT-PAGE RECORDS.

VAR I, J, K, L, NI, LAST: INTEGER;
CENDF ^ BOOLEAN;
#GLOBAL = W+

BEGIN
RESET(F); ENDF := FALSE;
REPEAT * NEW RECORD EVERY LOOP+
L := 1; NI := 1; Gi := TRUE;
LAST = F + LOC - 1;
W[I] := F + LOC; RELOCATABLE LOAD ADDRESS;
WHILE (L < 58) ^ (ENDF) A G DO
BEGIN
II := 0; L1 := L + 1; J1 := L W[J1] := 0;
REPEAT
IF F * LOC = LAST + 1 THEN LAST := LAST + 1
END;
END;
ELSE BEGIN G1=FALSE; GOTO 3 END;
IF F*.HRF>0 THEN MEMORY REFERENCE INSTR, USES TWO WORDS+ BEGIN
APPEND(W[1],3,5);
W[1]=F*.WHRF;
CASE F*.TYP OF
.. E8E = K=1; ECE = K=2
END;
APPEND(W[1],4,6);
END
ELSE =NO MEMORY REFERENCE+ BEGIN
W[1]=F*.WORD;
CASE F*.TYP OF
.. K=0;
K=1;
K=2;
K=3;
K=4;
END:
APPEND(W[1],3,6)
END
GET(F); IF EOF(F) THEN ENDF=TRUE;
UNTIL I=5*ENDF=1; I>59); I
APPEND(W[3],3+1,4);
N1=N1+1;
END;


=================================================================

PROCEDURE ENDRECORD:
VAR I,INTEGER* --GLOBALS-- W,TRNSFR
BEGIN
W[1]=20009; RECORD LENGTH=4, SHIFTED 8 BITS TO LEFT
W[4]=1;TRNSFR[3];
PUNCHRECORDE(END RECORDS,4)
END =ENDRECORD*;

=================================================================

PROCEDURE PREPAREOUTPUT;
PRINT TITLE PAGE WITH THE SERIAL NUMBER OF THE PROGRAM AND PROGRAM
TITLE, PROGRAMMER'S NAME, ETC. ALSO PUNCH A HEADER CARD WITH THE SERIAL
NUMBER ON IT
VAR I,J,01,02 : INTEGER; A#ALFA; C#ARRCHAR10;
PROCEDURE BLANKLINES(N:INTEGER);
VAR I:INTEGER;
BEGIN
  WRITE(EOL=E,E);
  FOR I=1 TO N DO
  BEGIN
    SPACE(37); OUTCH(E=E); SPACE(60); WRITE(E=E,EOL=E,E)
  END;
END BLANKLINES;

PROCEDURE CENTRE(A:ARRAY[0..99] OF CHAR; N1,N2:INTEGER);
VAR I,J,K,PAD:INTEGER;
BEGIN
  J=1; K=N2;
  WHILE A(J)=E AND (J<=N2) DO J=J+1;
  IF J>N2 THEN WHILE A(K)=E DO K=K-1;
  PADS=(K-J) DIV 2;
  OUTCH(E=E); SPACE(PADS); IF (K-J) MOD 2=0 THEN PADS=PADS-1;
  FOR I=J TO K DO OUTCH(A(I)); SPACE(PADS); OUTCH(E=E);
END CENTRE;

BEGIN
  D1=NPROG DIV 10; D2=NPROG MOD 10;
  *PUNCH HEADER CARDS
  PUNCHCARD(HEDCARD);
  PRINT TITLE PAGE;
  FOR I=1 TO 12 DO
  BEGIN
    SPACE(45); WRITE(MW[I],E,E); WRITE(PP[I],E,E);
    WRITE(DIG[(1+I)],E,E); WRITE(E,E,DIG[2*I+1],E,E)
  END;
  J=1; Repeat J=J+1; HONG(J+1)=INTEXT(J+7) Until (HONG(J)=E=J=61);
  HONG(J+1)=E;
  I=1; Repeat I=I+1 Until INTEXT(I)=E;
  I=I+1 Until I=80;
  FOR I=1 To 7 DO GET[I]=INTEXT[I];
  FOR I=8 To 16 DO GET[I]=E; PACK(C,1,A);
  IF A=MPASHT THEN ERROR(46);
  LINES[1]=SPACE(37); MULIPRINT(E,E,62); BLANKLINES(3);
  WRITE(*HEWLETT-PACKARD ASSEMBLY LANGUAGE PROGRAM
  BLANKLINES(3);
  WRITE(*HEWLETT-PACKARD ASSEMBLY LANGUAGE PROGRAM
  BLANKLINES(2); CENTRE(INTEXT,8,68); BLANKLINES(3);
  WRITE(*PROGRAMMER
  BLANKLINES(2); CENTRE(INTEXT,69,80); BLANKLINES(3);
  MULIPRINT(E,E,62); TITLE
  END PREPAREOUTPUT*;
==============================================================================

MAIN PROGRAM

BEGIN

* INITIALIZATION *

DIGITS = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9);  
CSETR = (15, 17, 19, 21, 23, 25, 27, 29);  
PSEUDOSUSET = (0, 5, 6, 7, 8, 9, 11, 12, 15, 18, 19, 20, 23, 24, 25, 26, 28, 31, 36);  
NONFATAL = (10, 12, 28, 33, 41, 42, 43, 45);  
NPROG1 = 0; WRITE(3); ENDINPUT = FALSE;

* NEW PROGRAM *

NPROG1 = NPROG1 + 1;

* RE-INITIALIZE VARIABLES *

REWRITE(S); WRITE(R); ENDINPUT = FALSE;

SINGLE = FALSE; EPAGE = FALSE; LSTFLAG = TRUE; FATAL = FALSE; EP1 = 0;  
TPLPRINT = FALSE; LSTOUTPUT = FALSE; PUNCHIN1 = FALSE; LISTINT = FALSE;  
AF1 = FALSE; RF1 = FALSE; OPTION1 = 0;  
PBI = 0; LII = C; LN1 = I; SNI = 1; ERI = 0; NER1 = C; LCI = 0; SCAN1 = TRUE;

FOR II = C TO S (DO SYSTAB[I].NAME = EMPTY;  
FOR II = 1 TO HLT DO LITAB[II, 1] = NOLI);

* PREPARE OUTPUT *

PRINT TITLE PAGE AND PUNCH HEADER CARD;  
PREPAREOUTPUT;

* READ AND INTERPRET THE CONTROL CARD *

READARRAY(INTEXT); IF ENDINPUT THEN GOTO 3;  
SQUEEZE(INTEXT, S*, LINE);  
S*, L1N1 = LN;  
S*, OPFLO = 0;  
S*, LOCFLD = 0;  
PUT(S);  
PRINTARRAY(INTEXT);  
LINES(2);  
COL1 = 1;  
GETNAME(A1);

IF A1 = ASHBE THEN ERROR(18);  
REPEAT  
IF CH IN PARAMETER THEN      
CASE CH OF      
BEGIN ASPROG1 = TRUE; AF1 = TRUE; LCI = 2; END;  
BEGIN ASPROG1 = FALSE; AF1 = TRUE; LCI = 0; END;

BEGIN AF1 = TRUE; LSTOUTPUT = TRUE;  
BEGIN AF1 = TRUE; PUNCHIN1 = TRUE;  
BEGIN AF1 = TRUE; LSTOUTPUT = TRUE;  
BEGIN AF1 = TRUE; PUNCHIN1 = TRUE;  
BEGIN AF1 = TRUE; LSTOUTPUT = TRUE;  
BEGIN AF1 = TRUE; PUNCHIN1 = TRUE;  
BEGIN AF1 = TRUE; LSTOUTPUT = TRUE;  
BEGIN AF1 = TRUE; PUNCHIN1 = TRUE;  
BEGIN AF1 = TRUE; LSTOUTPUT = TRUE;  
BEGIN AF1 = TRUE; PUNCHIN1 = TRUE;

EXE, EFE, EPI, END;  
BEGIN WRITE(3); PARAMETER C IS IGNORED;  
LINES(1); END;

END ELSE ERROR(19);  
NEXTCHAR(CH);  
UNTIL BLANKOREQD(CH);
LISTFLAG=LISTOUTPUT=BINOUTPUT=PUNCHBIN OR LISTBIN;
IF AF=OF THEN:
BEGIN:
ISP=FALSE; LC=0;
WRITE("PROGRAM ASSUMED TO BE RELOCATABLE *E*); LINES(2)
END:
IF "(LISTOUTPUT \ PUNCHBIN \ TBLPRINT \ LISTBIN) THEN ERROR(22);
IF NER=0 THEN:
BEGIN:
WRITE(ERROS IN CONTROL CARDS); LINES(1); MULTIPRINT("*E*",22);
LINES(2); PRINTER; SKIPTOEND; PUNCHCARD(ERRCARD); GOTO 2
END:
PASSONE:
IF TBLPRINT THEN "PRINT SYMBOL TABLE":
BEGIN:
TITLE: WRITE("SYMBOL TABLE:"); EOL,E03)
FOR i=0 TO SN DO:
BEGIN:
N=SYMPTR[i]:
WRITE("empty",SYMPTR[N].NAME);
IF "ASPROG THEN OUTCH(SYMPTR[N].TYP); OUTCH(3 E);;
OUTCH(SYMPTR[N].LOC); WRITE(EOL;E E);;
END:
END;
PASS TWO:
IF "(ERRPAGE < [ 1 ) THEN "PRINT PAGE NUMBERS WITH ERRORS":
BEGIN:
WRITE("ERRORS ON THE FOLLOWING PAGES"); LINES(2);
FOR i=1 TO EP NO:
IF I IN ERRPAGE THEN WRITE(I=5):
LINES(2);
END:
IF FATAL THEN BEGIN:
PUNCHCARD(ERRCARD):
WRITE("BINARY OUTPUT SUPPRESSED",EOL,E E); GOTO 2
END;
IF LISTBIN THEN TITLE:
IF BINOUTPUT THEN "BINARY OUTPUT REQUESTED":
IF ASPROG THEN ASPSTRING;
ELSE:
BEGIN "PUNCH/PRINT IN RELOCATABLE BINARY FORMAT":
NAME RECORD:
IF NT>0 THEN ENTRCORD:
IF XT>0 THEN EXTRCORD:
IF BCE>0 THEN OBLRECORDS(0,0):
IF EBC>0 THEN OBLRECORDS(1,1):
END:
END;
IF "ENDINPUT THEN "ASSEMBLE NEXT PROGRAM":
BEGIN WRITE(EOL;E13); GOTO 1 END;
LINES(2); WRITE("END OF RUNE"); WRITE(EOL);
END.
APPENDIX H

SAMPLE RUNS UNDER

THE CROSS ASSEMBLER
EXAMPLE OF A PROGRAM WITHOUT ERRORS

ASHEVILLE

- PROGRAM ASSUMED TO BE RELIABLE -

- NO ERRORS DETECTED IN PASS 1 -
EXAMPLE OF A PROGRAM WITHOUT ERRORS

SYMBOL TABLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>0.00006</td>
</tr>
<tr>
<td>INCA</td>
<td>X 0.00001</td>
</tr>
<tr>
<td>S2</td>
<td>L 0.00035</td>
</tr>
<tr>
<td>F32</td>
<td>M 0.00025</td>
</tr>
<tr>
<td>PCLT</td>
<td>A 0.00065</td>
</tr>
<tr>
<td>UITH</td>
<td>P 0.00066</td>
</tr>
<tr>
<td>HIST</td>
<td>L 0.00154</td>
</tr>
<tr>
<td>C1T</td>
<td>E 0.00164</td>
</tr>
<tr>
<td>P11X</td>
<td>X 0.00016</td>
</tr>
<tr>
<td>H1I</td>
<td>H 0.00032</td>
</tr>
<tr>
<td>TCH</td>
<td>A 0.00006</td>
</tr>
<tr>
<td>ETTP</td>
<td>A 0.00002</td>
</tr>
<tr>
<td>LTTP</td>
<td>0.07157</td>
</tr>
</tbody>
</table>
**EXAMPLE OF A PROGRAM WITHOUT ERRORS**

```
20 00035 00054 1:PLET OFF YOUTH
40 00124 00000 2:K.M. HSS 8
60 00124 00000 3:T.B. HSS 1
80 00157 17416 4:EXT 1'C..24...10R...10A.
100 00140 03100 5:NTH ASC 3'(FH,2)
50 END START
```

**5 NO ERRORS DETECTED IN PASS 1**
**Example of a Program Without Errors**

<table>
<thead>
<tr>
<th></th>
<th>PA'GF</th>
<th>PA'GFL</th>
<th>PA'GFL</th>
<th>PA'GFL</th>
<th>PA'GFL</th>
<th>PA'GFL</th>
<th>PA'GFL</th>
<th>PA'GFL</th>
<th>PA'GFL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PROGRAM WITH ERRORS

ERRORS DETECTED IN PASS 1

1) ERR. ASC: 41; ERROR IN PSEUDO-OP ASC: <1 CP I=44
2) ERR. IFN: IF followed by ELSE without an intervening END. THE SECOND PSEUDO-OP INSTRUCTION IS IGNORED
3) ERR. INV: ILLEGAL OPERATION
4) ERR. 100: CODE FOR LABEL IS NOT ONE OR TWO CHARACTERS
5) ERR. 101: CODE FOR LABEL IS NOT ONE OR TWO CHARACTERS

6) ERR. 107: ABOVE ERROR OCCURS IN A CODE OR IN I/H STATEMENT
7) ERR. 108: ABOVE ERROR OCCURS IN A LABEL OR ANY INSTRUCTION
8) ERR. 109: ABOVE ERROR OCCURS IN A LABEL OR ANY INSTRUCTION
9) ERR. 110: ABOVE ERROR OCCURS IN A LABEL OR ANY INSTRUCTION
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>00000</td>
</tr>
<tr>
<td>INC</td>
<td>00001</td>
</tr>
<tr>
<td>FL</td>
<td>00002</td>
</tr>
<tr>
<td>TFA</td>
<td>00003</td>
</tr>
<tr>
<td>SLC</td>
<td>00004</td>
</tr>
<tr>
<td>LTA</td>
<td>00005</td>
</tr>
<tr>
<td>LTA2</td>
<td>00006</td>
</tr>
<tr>
<td>TFA1</td>
<td>00007</td>
</tr>
<tr>
<td>TFA2</td>
<td>00008</td>
</tr>
<tr>
<td>DATA</td>
<td>00009</td>
</tr>
<tr>
<td>PUT1</td>
<td>00010</td>
</tr>
<tr>
<td>PUT2</td>
<td>00011</td>
</tr>
<tr>
<td>PUT3</td>
<td>00012</td>
</tr>
<tr>
<td>PUT4</td>
<td>00013</td>
</tr>
<tr>
<td>APP</td>
<td>00014</td>
</tr>
<tr>
<td>ASP</td>
<td>00015</td>
</tr>
</tbody>
</table>

Program with Errors
PROGRAM WITH ERRORS

ASPERITY
LINE FORMULA: DISTANCE, SLOPE, MID-POINT

```
START

*** FHRW: UNDECLARED VARIABLE

1. FHRW
2. FHRW
3. FHRW
4. FHRW
5. FHRW
6. FHRW
7. FHRW
8. FHRW
9. FHRW
10. FHRW
11. FHRW
12. FHRW
13. FHRW
14. FHRW
15. FHRW
16. FHRW
17. FHRW
18. FHRW

```

```
LINE FORMULI : DISTANCE., SLOPE., MIDDLE POINT.

-- INPUT THE FIRST TWO POINTS FOR THE DATA --

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>LCP</td>
</tr>
<tr>
<td>24</td>
<td>LPA 3</td>
</tr>
<tr>
<td>25</td>
<td>LPA 4</td>
</tr>
<tr>
<td>26</td>
<td>LPA 5</td>
</tr>
<tr>
<td>27</td>
<td>LPA 6</td>
</tr>
</tbody>
</table>

--- DECLARED VARIABLE ---

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>LPA DATA</td>
</tr>
<tr>
<td>31</td>
<td>LPA DATA</td>
</tr>
</tbody>
</table>

--- THE DISTANCE BETWEEN THE TWO POINTS ---

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>37</td>
</tr>
</tbody>
</table>

--- PRINT THE POINTS ---

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>JF</td>
</tr>
</tbody>
</table>

--- A VALID LITERAL ---

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>STR DATA</td>
</tr>
</tbody>
</table>

--- ERROR: LITERAL INCORRECT WITH THE INSTRUCTION USED ---

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>LPA DATA</td>
</tr>
</tbody>
</table>
**Line Formula 1: Distance, Slope, Mid-Point**

<table>
<thead>
<tr>
<th>LINE FORMULA 1</th>
<th>DISTANCE, SLOPE, MID-POINT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XI</strong></td>
<td><strong>X</strong></td>
</tr>
<tr>
<td><strong>X</strong></td>
<td><strong>XI</strong></td>
</tr>
</tbody>
</table>

**Fill the slope of the line**

- **LINE DATA**?
- **CPR**
- **EVAL**
- **IN**
- **PRINT**
- **NO-CALCULATION**

**Output the results**

- **SUM**
- **DIFF**
- **DIFF**
- **SUM**
- **DIFF**
- **SUM**

**Fill the intersection of the line**

- **LINE DATA**
- **ELE**
- **DIFF**
- **PRINT**
- **DIST**
- **BASE**
- **DIST**
- **BASE**
- **DIST**
- **BASE**
- **DIST**
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
<td>000115 016064 A</td>
<td>JSH FLOAT</td>
</tr>
<tr>
<td>107</td>
<td>00116 016013 A</td>
<td>FDP = F5</td>
</tr>
<tr>
<td>108</td>
<td>00120 104400</td>
<td>BST POINT+2</td>
</tr>
<tr>
<td>110</td>
<td>00127 01271 A</td>
<td>LEA = 2</td>
</tr>
<tr>
<td>111</td>
<td>00123 000400</td>
<td>CR4</td>
</tr>
<tr>
<td>112</td>
<td>00124 016064 P</td>
<td>LL4 = 32</td>
</tr>
<tr>
<td>113</td>
<td>00125 016065 X</td>
<td>JSH = J6</td>
</tr>
<tr>
<td>114</td>
<td>00126 016065 D</td>
<td>LF = F5</td>
</tr>
<tr>
<td>115</td>
<td>0127 008203</td>
<td>OCT 253.3584.278461.282</td>
</tr>
<tr>
<td>116</td>
<td>00133 000100</td>
<td>INC 64.654.2345/497441/2</td>
</tr>
<tr>
<td>117</td>
<td>00131 000100</td>
<td>ILLEGAL CHARACTER IN OPERAND FIELD</td>
</tr>
<tr>
<td>118</td>
<td>00123 000400</td>
<td>CR4</td>
</tr>
<tr>
<td>119</td>
<td>00124 016064 P</td>
<td>LL4 = 32</td>
</tr>
<tr>
<td>120</td>
<td>00127 01271 A</td>
<td>LEA = 2</td>
</tr>
<tr>
<td>121</td>
<td>00127 000400</td>
<td>CR4</td>
</tr>
<tr>
<td>122</td>
<td>00124 016064 P</td>
<td>LL4 = 32</td>
</tr>
<tr>
<td>123</td>
<td>00125 016065 X</td>
<td>JSH = J6</td>
</tr>
<tr>
<td>124</td>
<td>00126 016065 D</td>
<td>LF = F5</td>
</tr>
</tbody>
</table>

PAGE \textbf{7} \textbf{LINE FORMULI : DISTANCE, SLOPE, MID-POINT}
REFERENCES


