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

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

APPENDICES

LIST OF FIGURES

INTRODUCTION

I.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.

I.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

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.

I.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[3].

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.

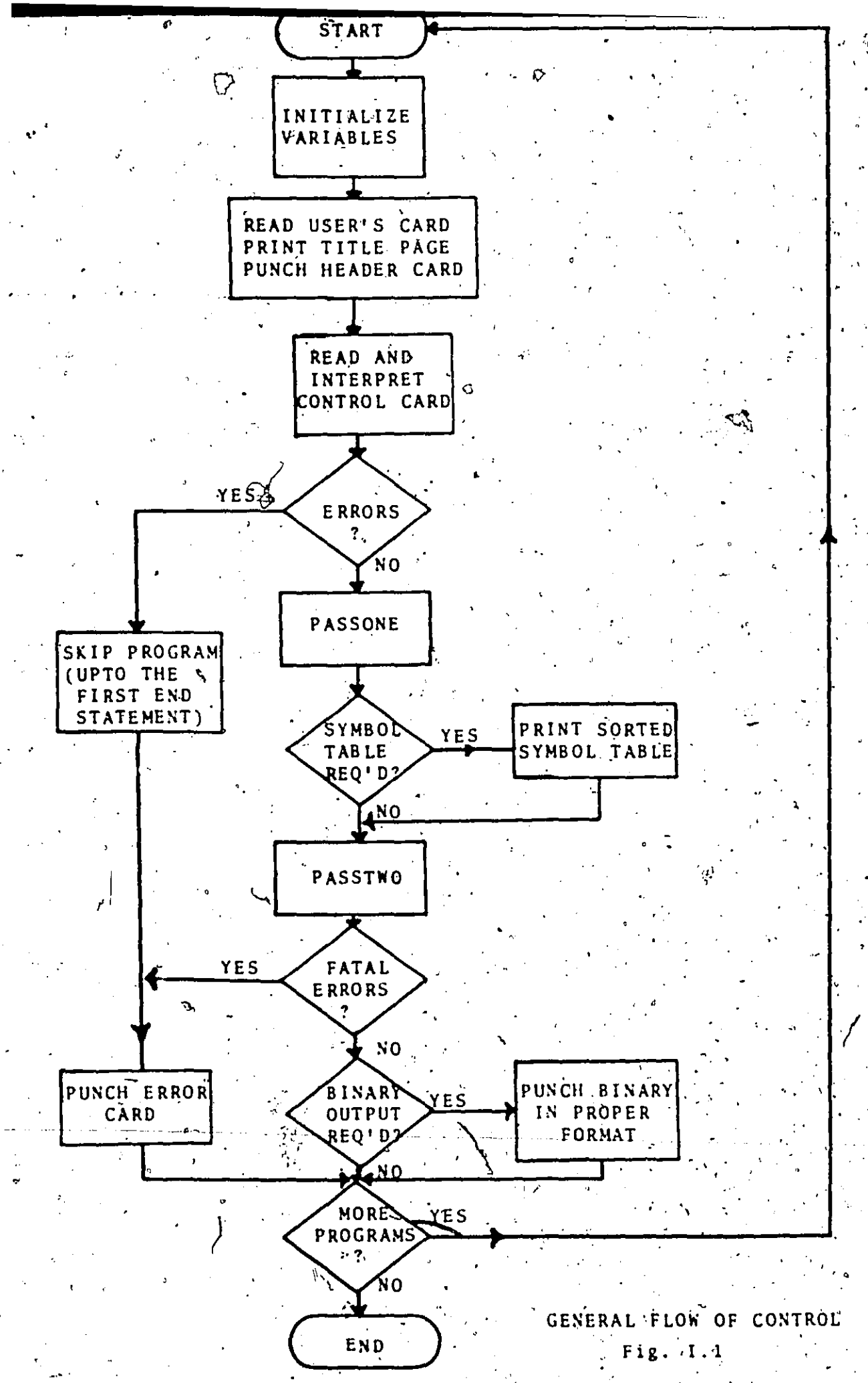
The instructions and control cards necessary for using the Cross-Assembler are described in Appendix A.

A general flow of control of the Cross-Assembler is shown in Figure I.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.



GENERAL FLOW OF CONTROL

Fig. I.1

field (ie. the location field) it is extracted and entered in the symbol table (using hash-coding), alongwith 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 processed 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 records 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.

I.4 HEWLETT-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 I.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-

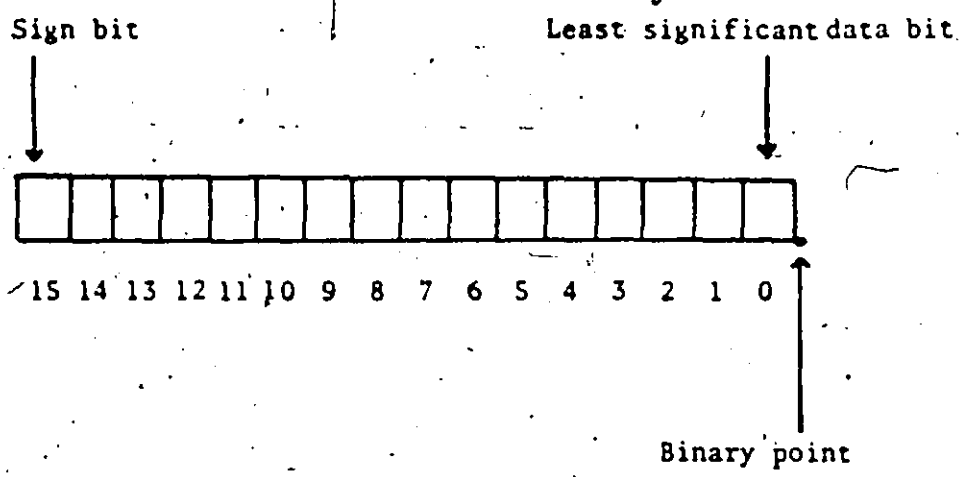


Fig. I.2

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 prowide 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 Manual[5].

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.

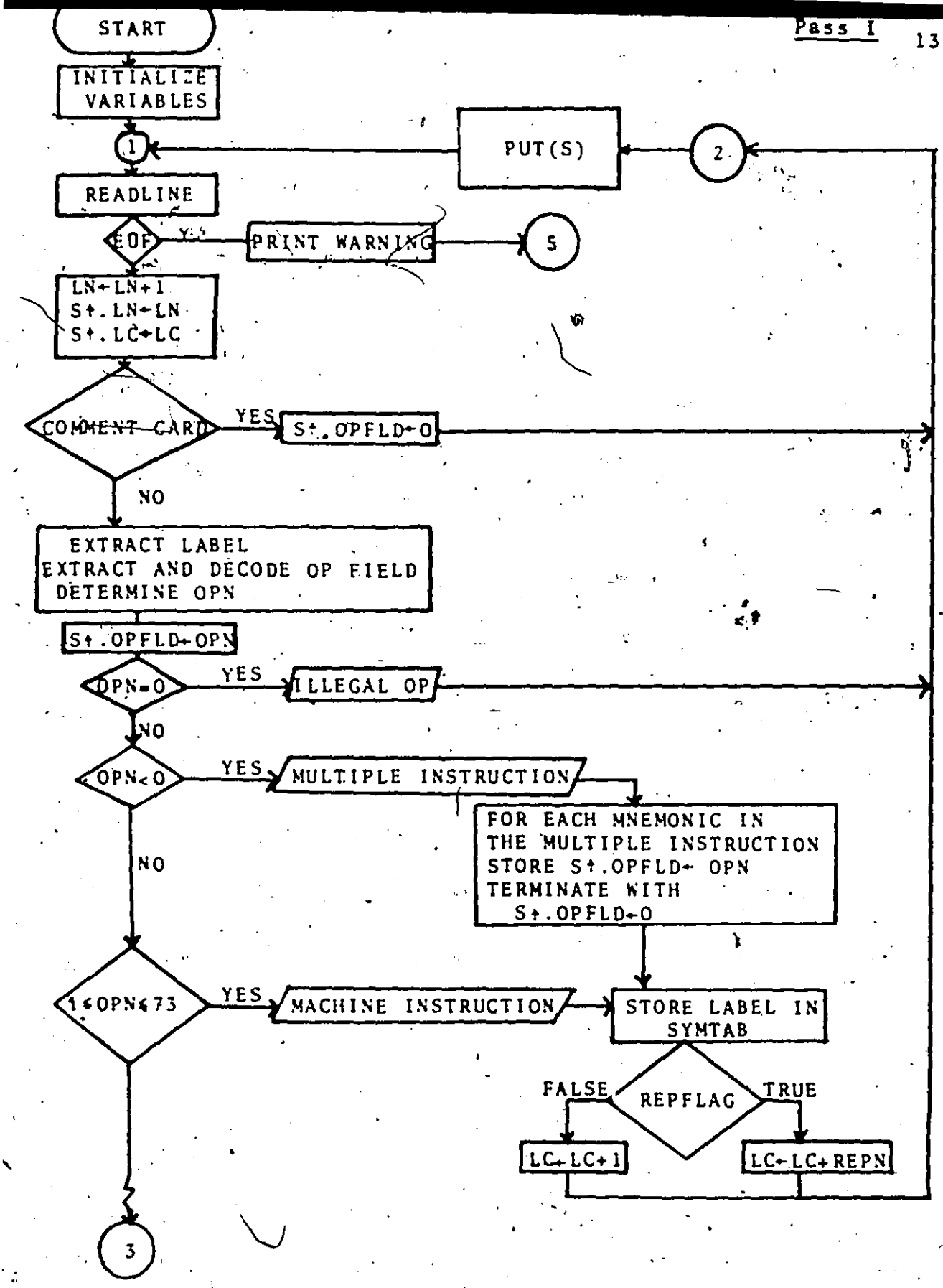


FIG. 1.1

FLOW-CHART FOR PASSONE

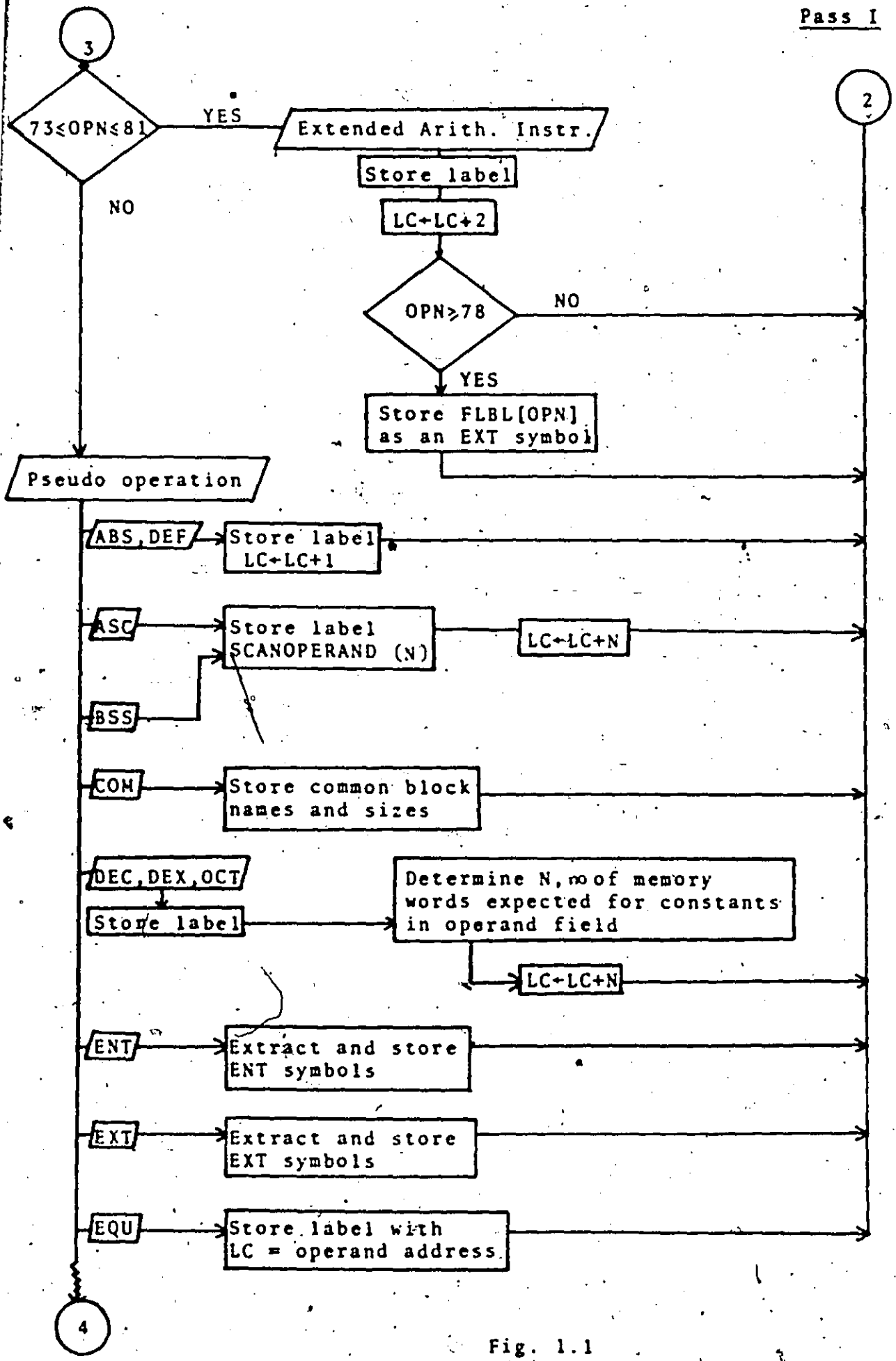

3
73≤OPN≤81
YES
NO
Extended Arith. Instr.
Store label
LC←LC+2
OPN≥78
NO
YES
Store FLBL[OPN] as an EXT symbol
2
Pseudo operation
ABS,DEF
Store label LC←LC+1
ASC
Store label SCANOPERAND (N)
LC←LC+N
BSS
COM
Store common block names and sizes
DEC,DEX,OCT
Store label
Determine N, no of memory words expected for constants in operand field
LC←LC+N
ENT
Extract and store ENT symbols
EXT
Extract and store EXT symbols
EQU
Store label with LC = operand address
4

Fig. 1.1
(cont.)

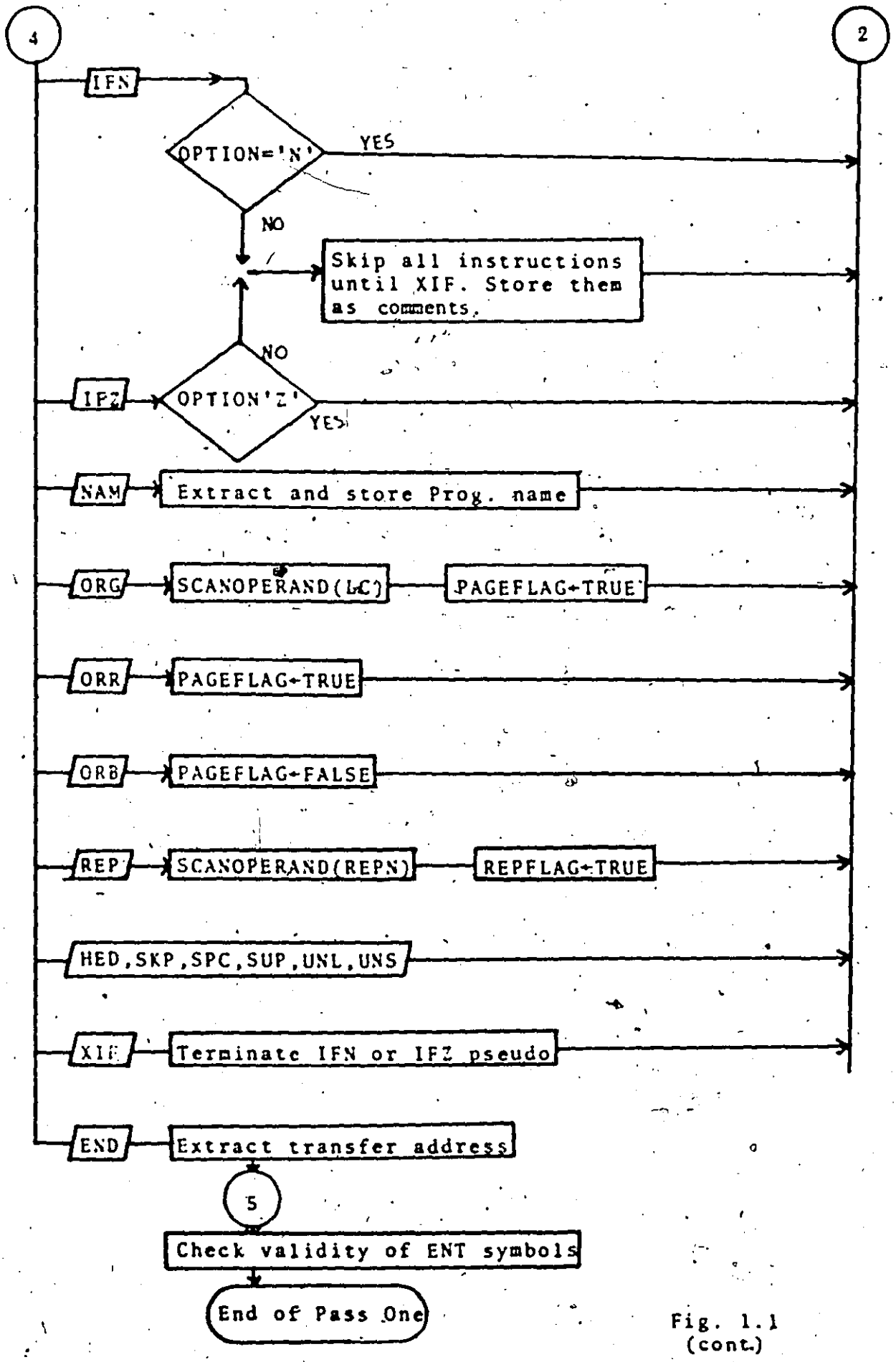

4
2
IFN
OPTION='N'
YES
NO
Skip all instructions until XIF. Store them as comments.
NO
IFZ
OPTION'Z'
YES
NAM
Extract and store Prog. name
ORG
SCANOPERAND(LC)
PAGEFLAG←TRUE
ORR
PAGEFLAG←TRUE
ORB
PAGEFLAG←FALSE
REP
SCANOPERAND(REPN)
REPFLAG←TRUE
HED,SKP,SPC,SUP,UNL,UNS
XIF
Terminate IFN or IFZ pseudo
END
Extract transfer address
5
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[3]; 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[6] technique. The following scheme is used:

H=((ABS(ORD(NAME)) DIV 4096) MOD SZ

where H= hash-address

NAME= Symbol, as an alfa variable

ORD(NAME)= Internal representation of NAME on CDC as an integer (left-justified),

SZ= size of symbol table (an odd number).

All the 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,..... 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, alongwith 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*. 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 upto five alpha-numeric 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 ($1 \leq OPN \leq 73$) 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 (eg. storing the label and card-image etc.) described earlier.

Multiple Instruction

If the operation field contains a number of operation mnemonics, separated by comma , then the value of OPN for each of these mnemonic 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 ($74 \leq OPN \leq 81$) 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 (alfa 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 ← EMPTY and NMN ← 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 out 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 PAGEFLAGE ← 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, or 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 to be 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 ← 0 (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, alongwith 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 '.' 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 (SY.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)
= 1 (if label is base-page relocatable)

TRNSFR [2]= 1 (to indicate a non-blank label)

TRNSFR [3]= value of location counter for the label.

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 ENDLC. 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

TYP	(char)	relocatability of the operand, viz.
		=R, program relocatable,
		=B, base page relocatable
		=C, common relocatable
		=X, external
		=blank, absolute
MREF	(integer)	contains the operand address if the operand is a memory-reference instruction; otherwise MREF = -1

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.

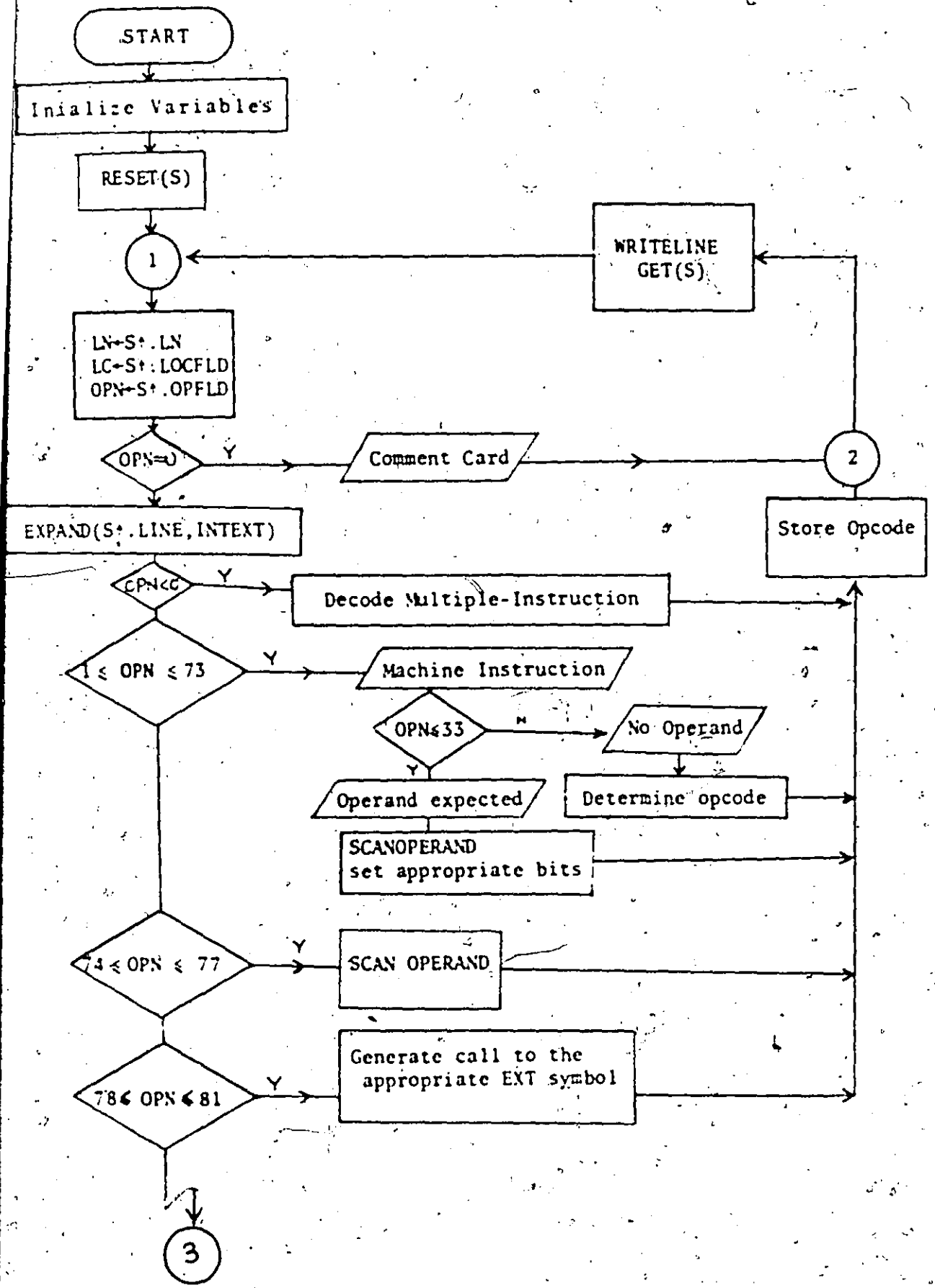


Fig. 2.1 FLOW CHART FOR PASSTWO

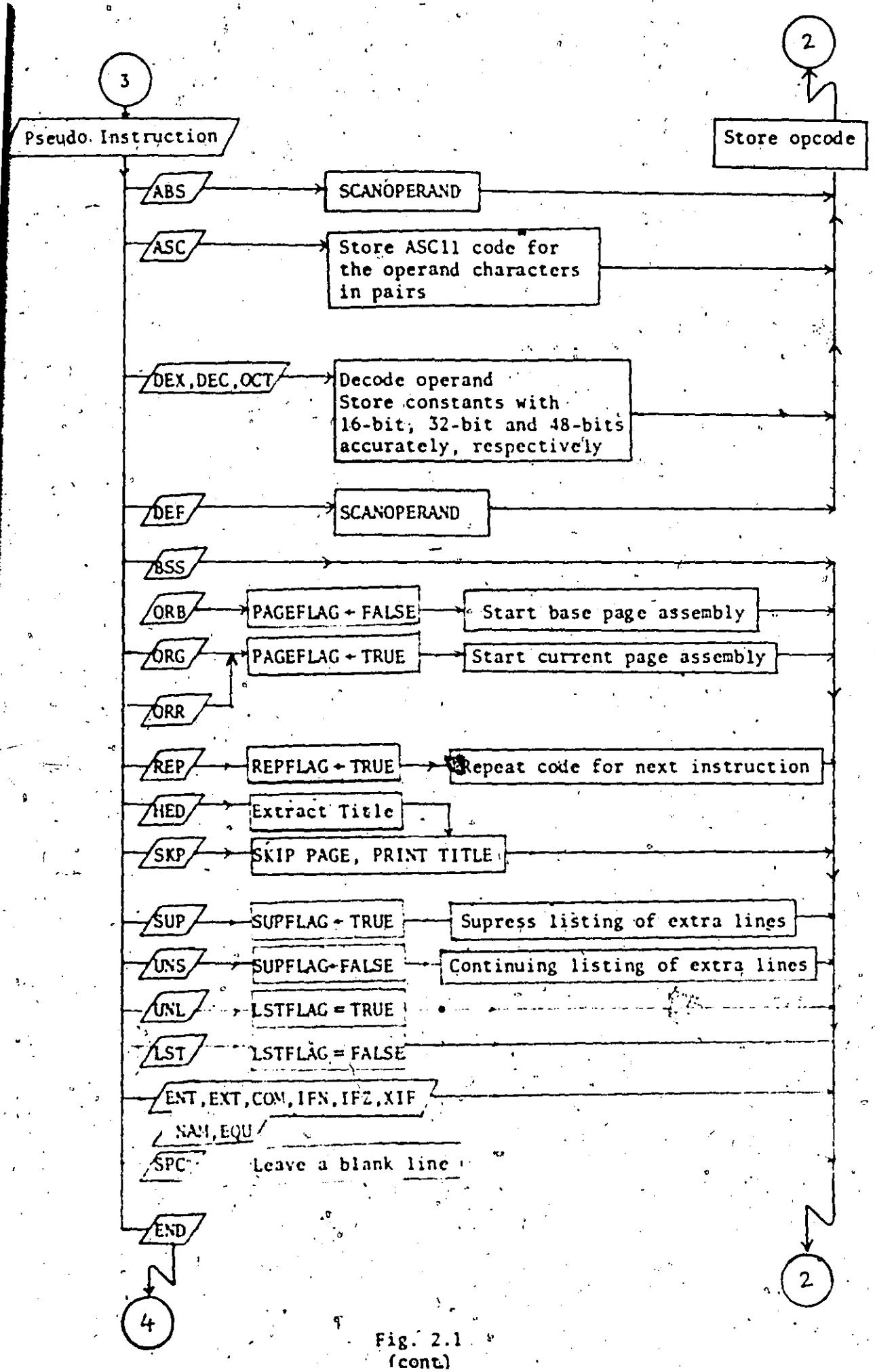

3
Pseudo Instruction
2
Store opcode
ABS
SCANOPERAND
ASC
Store ASCII code for the operand characters in pairs
DEX,DEC,OCT
Decode operand
Store constants with 16-bit, 32-bit and 48-bits accurately, respectively
DEF
SCANOPERAND
BSS
ORB
PAGEFLAG ← FALSE
Start base page assembly
ORG
PAGEFLAG ← TRUE
Start current page assembly
ORR
REP
REPFLAG ← TRUE
Repeat code for next instruction
HED
Extract Title
SKP
SKIP PAGE, PRINT TITLE
SUP
SUPFLAG ← TRUE
Supress listing of extra lines
UNS
SUPFLAG←FALSE
Continuing listing of extra lines
UNL
LSTFLAG = TRUE
LST
LSTFLAG = FALSE
ENT,EXT,COM,IFN,IFZ,XIF
NAM,EQU
SPC
Leave a blank line
END
4
2

Fig. 2.1
(cont)

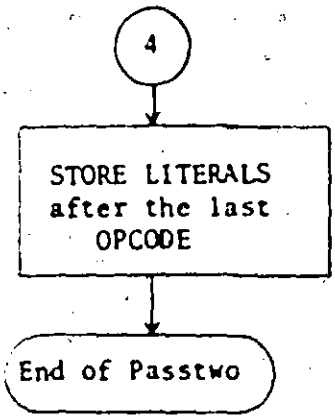


Fig. 2.1
(cont.)

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 (S↑.LOCFLD) is stored in LC, that of the operation number (S↑.OPFLD) is stored in OPN and the compressed card image (S↑.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 (S↑.LN) and the card-image (S↑.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, eg. 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

```
[ ALS ]                                  [ ALS ]
[ ARS ]                                  [ ARS ]
[ RAL ]                                  [ RAL ]
[ RAR ]                                  [ RAR ]
[ ALR ]   [,CLE]    [,SLA]    ,        [ ALR ]
[ ALF ]                                  [ ALF ]
[ ERA ]                                  [ ERA ]
[ ELA ]                                  [ ELA ]
```

B-Register type

```
[ BLS ]                                  [ BLS ]
[ BRS ]                                  [ BRS ]
[ RBL ]                                  [ RBL ]
[ RBR ]                                  [ RBR ]
[ BLR ]   [,CLE]    [,SLB]    ,        [ BLR ]
[ BLF ]                                  [ BLF ]
[ ERB ]                                  [ ERB ]
[ ELB ]                                  [ ELB ]
```

Fig. 2.2 (a)

MULTIPLE INSTRUCTIONS (Shift-Rotate Group)

A-Register type

```
[CLA]          [CLE]
[CMA] [,SEZ]   [CME]  [,SSA]  [,SLA]  [,INA]  [,SZA]  [,RSS]
[CCA]          [CCE]
```

B-Register type

```
[CLB]          [CLE]
[CMB] [,SEZ]   [CME]  [,SSB]  [,SLB]  [,INB]  [,SZB]  [,RSS]
[CCB]          [CCE]
```

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

etc.

We shall not discuss the usage or usefulness of a multiple-instruction here. The user should refer to the HP Assembler Manual[5].

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 ($1 \leq OPN \leq 73$) refers to a machine instruction. They may be divided into two groups.

The first group of instructions ($1 \leq OPN \leq 33$) 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[3] terminology)

appearing in an operand field must be compatible; base-page, current-page and common-relocatable symbols cannot be intermixed. 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):

TYP = R, program relocatable
= B, base-page relocatable
= C, common relocatable
= X, external reference
= blank, absolute (non-relocatable)

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 $1 \leq OPN \leq 14$ (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 ($34 \leq OPN \leq 73$) 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 instrctions 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 appropirate 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

ASC n, ⟨string of 2n characters⟩

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 PASSONE, 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 encounterd, which changes the contents of HDNG.

UNL: Normally, each statement of the program is listed alongwith 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 (initally 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, ie. 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  =FO.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[I,1] = binary form of the Ith literal

LITTAB[I,2] = the address of the Ith 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
  - A - two ASCII characters
  - 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[7]. 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

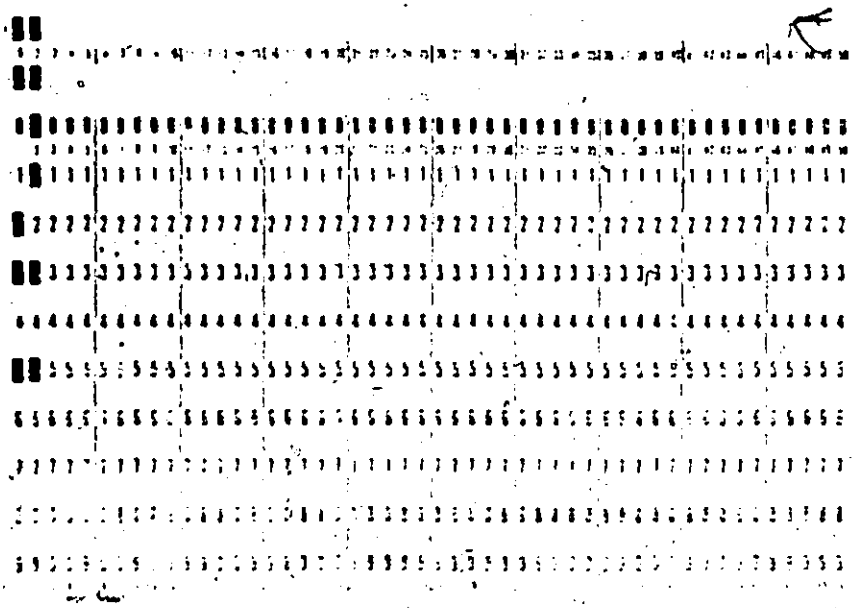
For the absolute programs each machine word (16-bits) is broken up into two parts and each half-word (8-bits) is punched in a card column in the punch position 12 to 5. For example, the 16-bit binary number

1 011 001 110 101 111

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 upto 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],---,W[LAST-1]= Machine-code words,

W[LAST] = CHECKSUM (i.e. Arithmetic sum of all the words in the record except the first and last, modulu 200000 octal)

The maximum number of words in a record (ie. 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:

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

ENT record - contains upto 14 entry-point symbols and their addresses.

EXT record - contains upto 19 external symbols and their ID numbers.

DBL record - contains upto 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 NAM record, a number of DBL records and an END record, and optionally, an ENT record and a number of EXT records.

The NAM, ENT, EXT and END records are rather straight forward 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 those 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.[8]

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 alfa 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 auther 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.                          NAME
ATTACH(PASCAL,ID=.........)
ATTACH(HPASMB,ID=.........)
PASCAL(LOAD=HPASMB)
DISPOSE(P,*PS)
End of Record
HPASMB.          Program Title (Cols 8-68)   Name(Cols 69-80)
ASMB,R,L,N,.....etc.   (See Section C)
   .
   .
(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 upto 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
} only one is required

L - produce program listing
B - produce punched binary output
T - produce symbol table
Q - print binary records+
} at least one is required

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 'ERRO1' 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 upto 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 sift 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
COM	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 percision 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)

FMP	Floating multiply
FSB	Floating subtract
HED	Print heading at top of each page
HLT	Halt
IFN	When N appears in Control Statement, assemble ensuing instructions.
IEZ	When Z appears in Control Statement, assemble ensuing instructions
INA	Increment A by 1
INB	Increment B by 1
IOR	Inclusive "or" to A
ISZ	Increment, then skip if zero
JMP	Jump
JSB	Jump to subroutine
LDA	Load into A
LDB	Load into B
LIA	Load into A from I/O channel
LIB	Load into B from I/O channel
LSL	Logical long shift left
LSR	Logical long shift right
LST	Resume list output (follows a UNL)
MIA	Merge (or) into A from I/O channel
MIB	Merge (or) into B from I/O channel
MPY	Muliply
NAM	Names relocatable program

ALPHABETIC LIST OF INSTRUCTIONS (cont)

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

ALPHABETIC LIST OF INSTRUCTIONS (cont)

SSA	Skip if sign A = 0
SSB	Skip if sign B = 0
STA	Store A
STB	Store B
STC	Set I/O control bit
STF	Set I/O flag
STO	Set overflow bit
SUP	Suppress list output of additional code lines
SWP	Switch the (A) and (B)
SZA	Skip if A = 0
SZB	Skip if B ≠ 0
UNL	Suppress list output
UNS	Resume list output of additional code lines
XIF	Terminate an IFN or IFZ group of instructions
XOR	Exclusive "or" to A

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
BRS
RAL
RBL
RAR
RBR
ALR
BLR
ERA
ERB
ELA
ELB
ALF
BLF
SLA
SLB

No-operation

NOP

Alter-skip

CLA
CLB
CMA
CMB
CCA
CCB
CLE
CME
CCE
SEZ
SSA
SSB
INA
INB
SZA
SZB
SLA
SLB
RSS

GROUP 3 INPUT/OUTPUT, OVERFLOW AND HALT INSTRUCTIONS

Input/Output

STC
CLC
LIA
LIB
MIA
MIB
OTA
OTB
STF
CLF
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.
14. Illegal mixing of SRG and ASG instructions.

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 and 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.
24. Absolute 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.

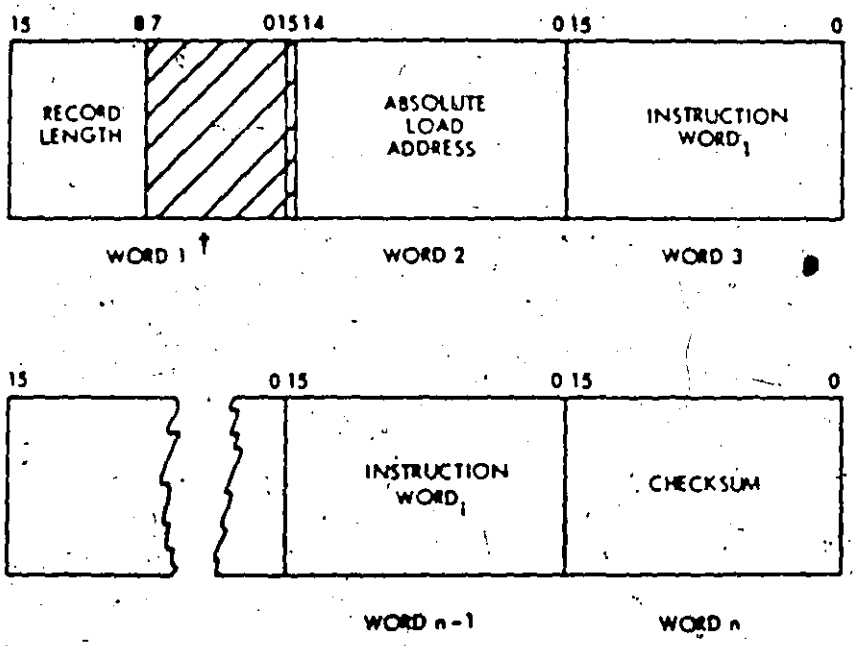
C. Non-Fatal Errors

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.
9. Operand exceeds 77777 octal.
10. An IFN or an IFZ follows either an IFN or an IFZ without an intervening XIF. The second pseudo is ignored.

APPENDIX E

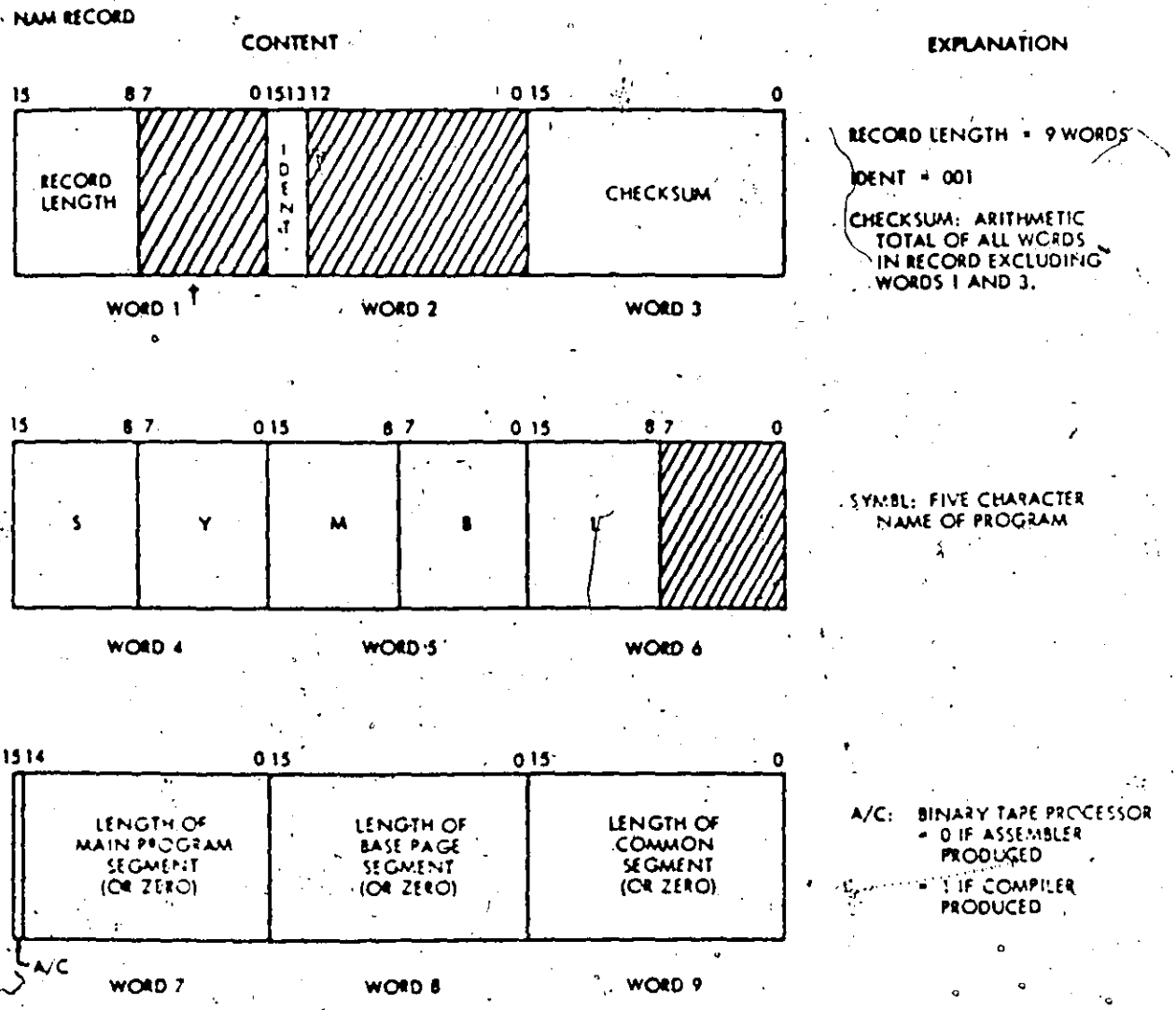
ABSOLUTE RECORD FORMAT



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

APPENDIX F

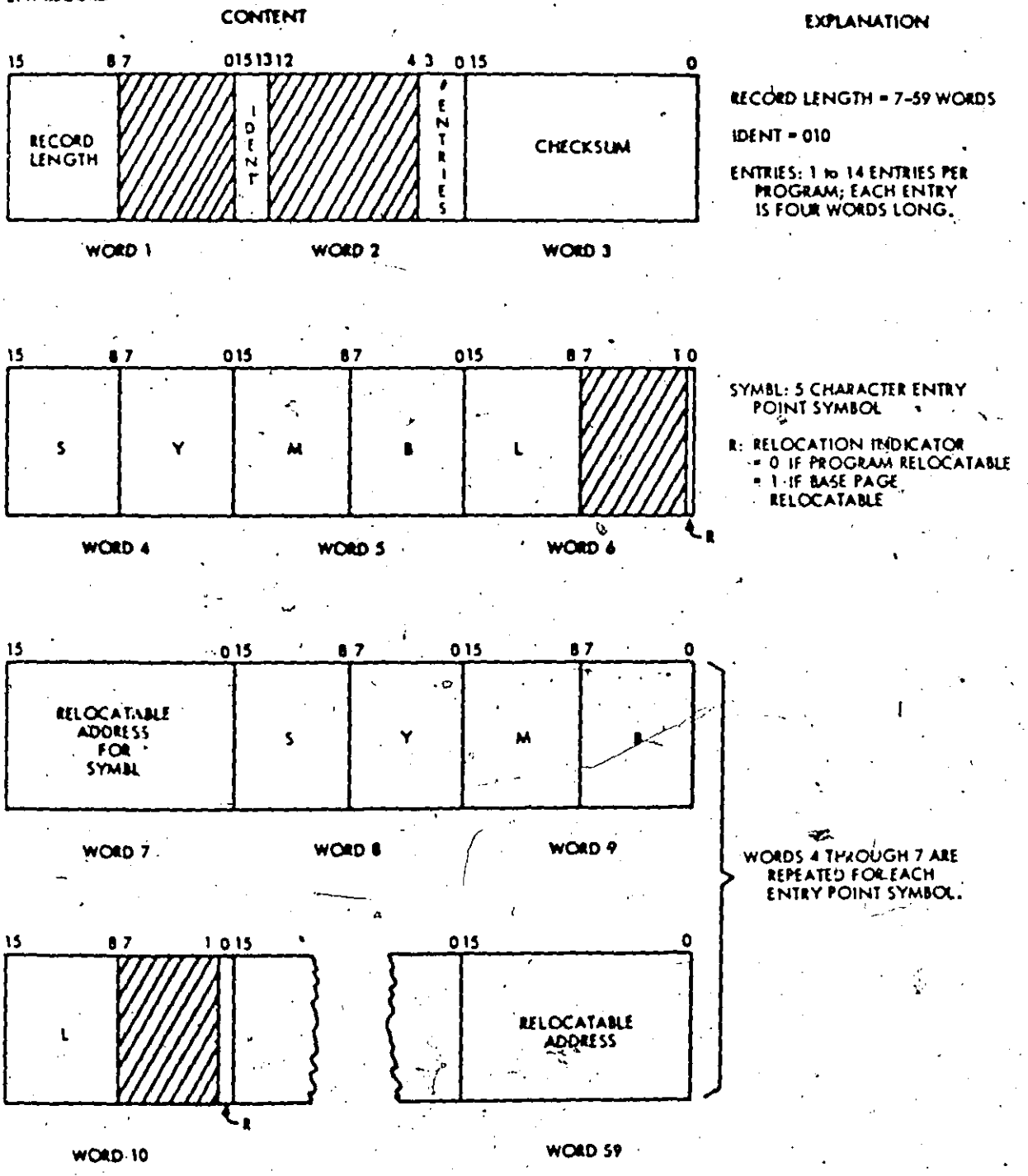
RELOCATABLE RECORD FORMAT



NAM RECORD

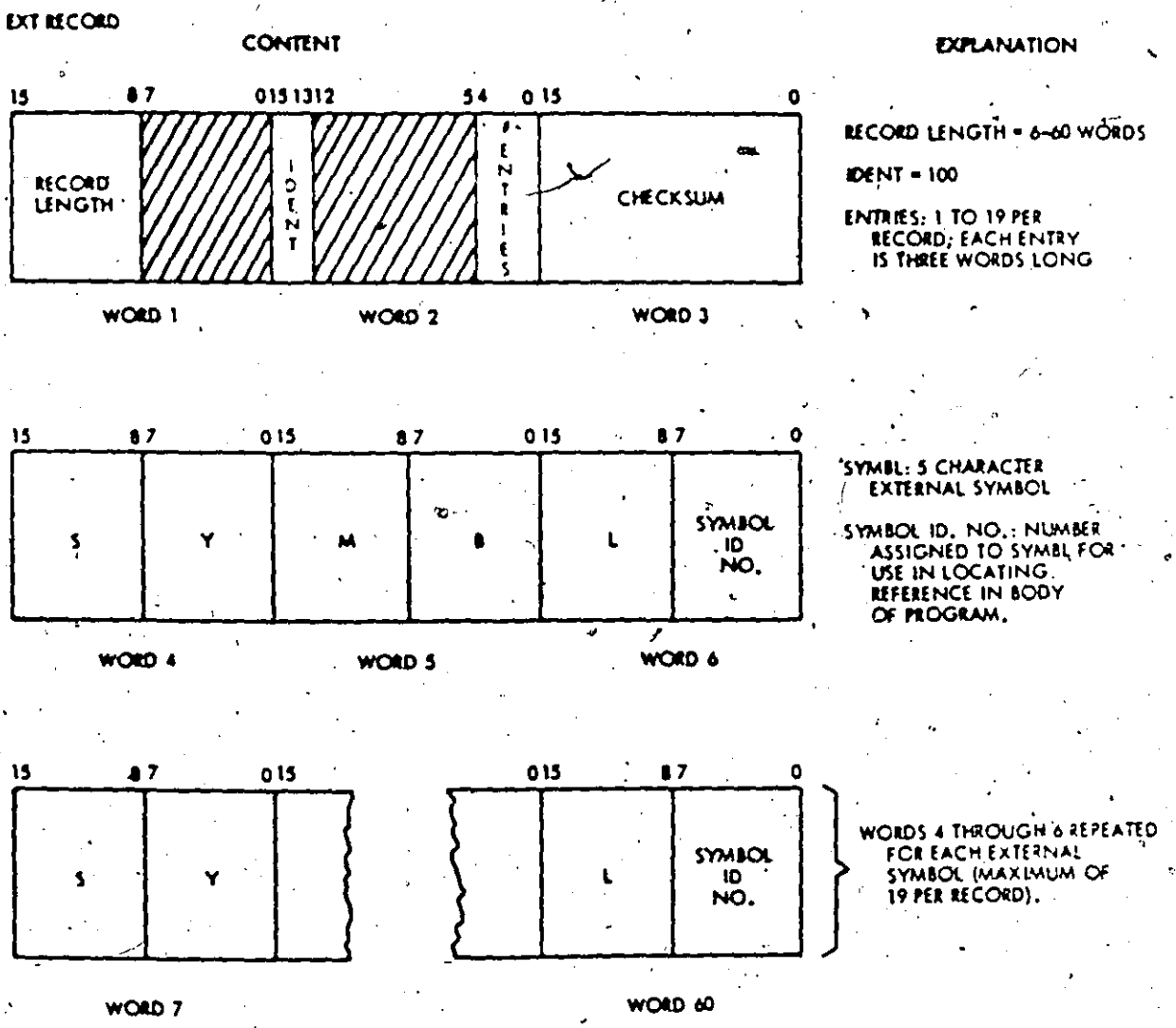
RELOCATABLE RECORD FORMAT (continued)

ENT RECORD



ENT RECORD

RELOCATABLE RECORD FORMAT (continued)

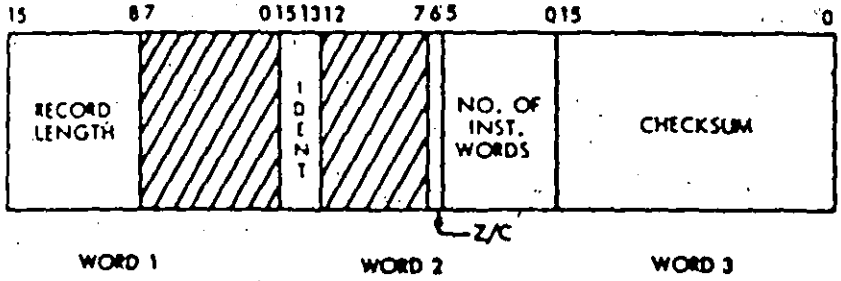


EXT RECORD

DBL RECORD

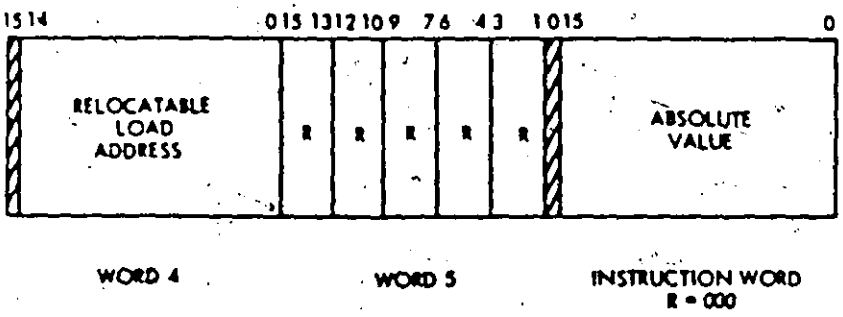
CONTENT

EXPLANATION



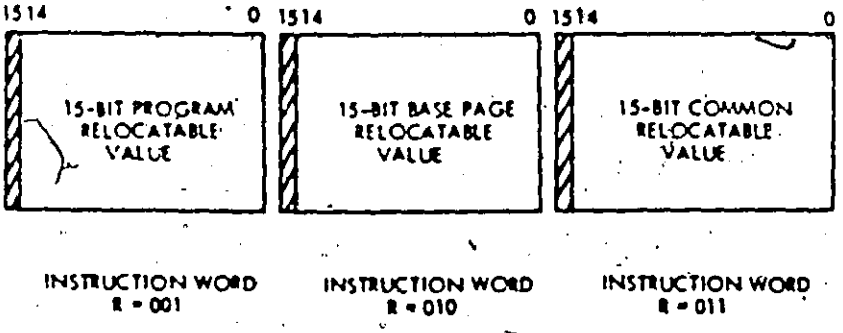
RECORD LENGTH = 5-60 WORDS
IDENT = 011
Z/C: BASE/CURRENT PAGE LOADING
= 0 FOR BASE PAGE
= 1 FOR CURRENT PAGE

NO. OF INST. WORDS: 1 TO 45 LOADABLE INSTRUCTION WORDS PER RECORD



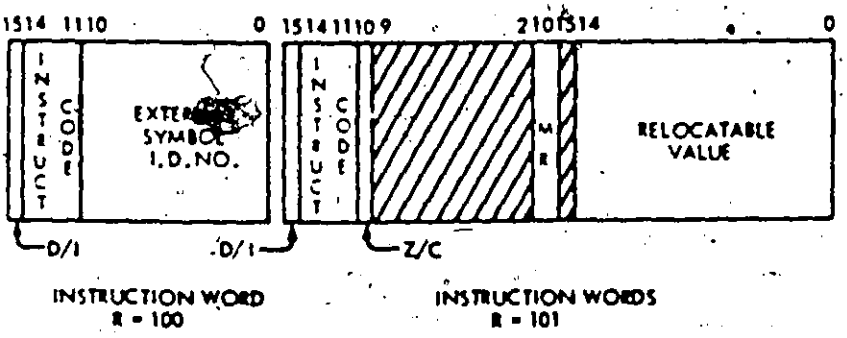
RELOCATABLE LOAD ADDRESS: STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW.

R's: 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



R_1 IS RELOCATION INDICATOR FOR INSTRUCTION WORD1; R_2, FOR INSTRUCTION WORD2; ETC. MEMORY REFERENCE INSTRUCTIONS USE TWO WORDS, WITHIN THE TWO-WORD GROUP, "MR" INDICATES RELOCATABILITY OF OPERAND SPECIFIED IN SECOND WORD:

00 = PROGRAM RELOCATABLE
01 = BASE PAGE RELOCATABLE
10 = COMMON RELOCATABLE



D/I: INDIRECT ADDRESSING

0 = DIRECT
1 = INDIRECT

Z/C: BASE/CURRENT PAGE LOCATION OF OPERAND ADDRESS AS DETERMINED BY LOADER.

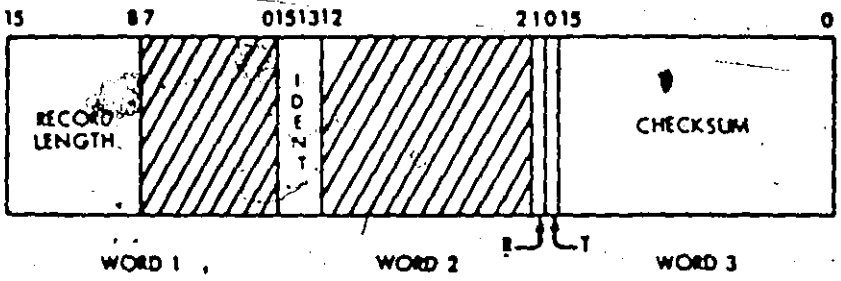
0 = BASE PAGE
1 = CURRENT PAGE

RELOCATABLE RECORD FORMAT (continued)

END RECORD

CONTENT

EXPLANATION

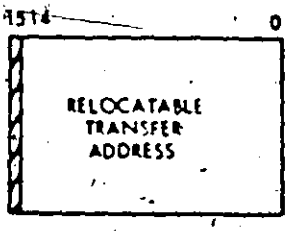


RECORD LENGTH = 4 WORDS

IDENT = 101

R: RELOCATION INDICATOR FOR TRANSFER ADDRESS

- = 0 IF PROGRAM RELOCATABLE
- = 1 IF BASE PAGE RELOCATABLE



T: TRANSFER ADDRESS INDICATOR

- = 0 IF NO TRANSFER ADDRESS IN RECORD
- = 1 IF TRANSFER ADDRESS PRESENT

END RECORD

APPENDIX G

PROGRAM LISTING OF THE CROSS-ASSEMBLER

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

```
{********** H.P. CROSS-ASSEMBLER, VERSION C, AUGUST 1974 **********}
{ AUTHOR: K.AHMED, DEPT. APPLD. MATHS., MCMASTER UNIV., HAMILTON, ONT }
{ PASCAL COMPILE OPTIONS :          $A-,D-,I-,R+,X-,C-}

CONST
   LP = 58             { NO. OF LINES PER OUTPUT PAGE}
   MLT =25;            { SIZE OF LITERAL TABLE }
   SZ = 499;           { SIZE OF SYMBOL TABLE}
   MXER = 5;           { MAX NO. OF ERRORS LISTED PER LINE}
   MNT = 14;           { MAX. NO. OF ENTRY SYMBOLS }
   MXT = 20 ;          { MAX. NO. OF EXTERNAL SYMBOLS }
   NO = 6;             { NO. OF DIGITS TO BE PRINTED BY OUTOCT}
   NI = 139;           { NO. OF INSTRUCTIONS }
   NOLIT = -1;         { FOR INITIALIZING THE LITERAL TABLE }
   EMPTY = =             =;

TYPE
   BIT = 0..1;
   ARRINT3  = ARRAY[1..3] OF INTEGER;
   ARRINT16 = ARRAY[1..16] OF INTEGER;
   ARRALFA9 = ARRAY[1..9] OF ALFA;
   ARRCHAR10 = ARRAY[1..10] OF CHAR;
   ARRCHAR96 = ARRAY [1..96] OF CHAR;
   INDEXN = 1..NI;
   INDEXS = 0..SZ;
   PRECISION = (SINGLE,DOUBLE);
   STATEMENT = RECORD
                  LN,LOCFLD,OPFLD : INTEGER;
                  LINE:ARRALFA9
               END;
   OUTCODE = RECORD
                LOC,WORD,PREF:INTEGER; TYP:CHAR;
             END;
   FILEOUTCODE = FILE OF OUTCODE;

VAR
   EP, PC, LI, SN, NT, MP, ER, NER, NPROG, NMN : INTEGER;
   WORD,CODE,ENDLC,LN,NLN,SRC,KIND : INTEGER;
   I,J,LC,BPC,COL,NUM,OPN,N, LOC,XT,COM,REPN : INTEGER;

   CH, TYP, T, OPTION : CHAR;
   LBL,PROG,A : ALFA;
   X,MAXREAL,MINREAL,EPSILON : REAL;

   AF, PF, ENDINPUT, FIRSTERR, LSTOUTPUT : BOOLEAN;
   ERRFLAG, INDIRECT, REPFLAG, PUNCHBIN, TRLPRINT : BOOLEAN;
   SCAN1, SCANTOCOMMA, LISTBIN, BINOUTPUT, FATAL : BOOLEAN;
   ABSPROG, PRESENT, LSTFLAG, SUPFLAG, PAGEFLAG, EXTRALINE : BOOLEAN;

   CSET : SET OF   15..36;
   PSEUDOSUBSET : SET OF    0..23;
   PARAMETER, LITCHARS, ALPHA, DIGITS : SET OF   CHAR;
   NONFATAL,ERRPAGE : SET OF   0..58;

   R,B : FILEOUTCODE;
   S : FILE 12 OF STATEMENT;
   P(OUT) : FILE OF INTEGER;
```

```
  OPTABLE: ARRAY[INDEXN] OF ALFA;
  OPNUM : ARRAY[INDEXN] OF INTEGER;
  OPCODE : ARRAY[1..77] OF INTEGER;
  OPCODE2 : ARRAY[39..57] OF INTEGER;
  INVERT : ARRAY[...255] OF INTEGER;
  O : ARRINT3;
  JT, KT : ARRAY[1..8] OF INTEGER;
  PT : ARRAY [55..73] OF INTEGER;
  COMP : ARRAY[1..16] OF INTEGER;
  FLAL : ARRAY[78..81] OF ALFA;
  ASCIICODE : ARRAY[CHAR] OF INTEGER;
  HH,PP : ARRAY [1..12] OF ALFA;
  DIG : ARRAY [0..9,1..12] OF ALFA;
  IMAGE : ARRAY[0..19] OF INTEGER;
  HEDCARD, ERRCARD : ARRINT16;
  ENT : ARRAY [1..MNT] OF RECORD
                            SYM:ALFA; LOC:INTEGER; TYP:CHAR
                          END;
  EXT : ARRAY[1..MXT] OF ALFA;
  LITTAB : ARRAY [1..MLT,1..2] OF INTEGER;
  SYMPTR : ARRAY[INDEXS] OF INTEGER;
  SYMTAB : ARRAY[INDEXS] OF
       PACKED RECORD
            NAME:ALFA; LOC:INTEGER; TYP:CHAR
       END;

  TRNSFR : ARRINT3;
  INTEXT,HDNG : ARRCHAR80;
  W : ARRAY [1..60] OF INTEGER;
  ERRSTORE : ARRAY[1..MXER] OF INTEGER;

VALUE
  MAXREAL = 211745473231241324168;      {MAX. REAL CONSTANT FOR HP}
  MINREAL = 152165343734732517078;      {MIN. REAL CONSTANT FOR HP}
  EPSILON = 164640000000000000008;      {A SMALL CONSTANT FOR ROUNDING} *

  {TABLE OF INSTRUCTIONS (ALPHABETICAL)}
   OPTABLE = ('ABS ','ADA ','ADB ','ALF ','ALR ','ALS ','AND ','ARS ','ASC ', *
              'ASL ','ASR ','BLF ','BLR ','BLS ','BRS ','BSS ','CCA ','CCB ', *
              'CCE ','CLA ','CLB ','CLC ','CLE ','CLF ','CLO ','CMA ','CMB ', *
              'CME ','COM ','CPA ','CPB ','DEC ','DEF ','DEX ','DIV ','DLD ', *
              'DST ','ELA ','ELB ','END ','ENT ','EQU ','ERA ','ERB ','EXT ', *
              'FAD ','FDV ','FMP ','FSB ','HED ','HLT ','IFN ','IFZ ','INA ', *
              'INB ','IOR ','ISZ ','JMP ','JSB ','LDA ','LDB ','LIA ','LIB ', *
              'LSL ','LSR ','LST ','MIA ','MIB ','MPY ','NAM ','NOP ','OCT ', *
              'ORB ','ORG ','ORR ','OTA ','OTB ','PAL ','PAR ','PBL ','PBR ', *
              'REP ','REL ','RRR ','RSS ','SEZ ','SFC ','SFS ','SKP ','SLA ', *
              'SLB ','SOC ','SOS ','SPC ','SSA ','SSB ','STA ','STB ','STC ', *
              'STF ','STO ','SUP ','SZA ','SZB ','UNL ','UNS ','XIF ', *
              'XOR ');

  {TABLE THAT ARRANGES INSTRUCTIONS BY GROUPS}
  OPNUM =    ( 82,   1,   2,  44,  43,  39,   3,  40,  83,  28,  29, *
               52,  51,  47,  48,  84,  60,  70,  66,  58,  68,  15, *
               55,  16,  34,  53,  69,  65,  85,   4,   5,  86,  87, *
               88,  75,  76,  77,  46,  54,  89,  90,  91,  45,  53, *
               92,  78,  81,  80,  79,  93,  17,  94,  95,  82,  72, *
                6,   7,   8,   9,  10,  11,  18,  19,  30,  31,  96, *
               20,  21,  74,  97,  38,  98,  99, 100, 101,  22,  23, *
               41,  42,  49,  50, 102,  32,  33,  67,  64,  24,  25, *
              103,  56,  57,  35,  36, 104,  61,  71,  12,  13,  26, *

               27,  37, 105, 109,  63,  73, 106, 107, 108, 141);
```

```
*OPCODE FOR MACHINE INSTRUCTIONS (BY GROUPS)*
OPCODE =    ( 400600B,   440000B,   100000B,   500000B,   540000B,   300000B,
              340000B,   240000B,   140000B,   600000B,   640000B,   700000B,
              740000B,   200000B,   160700B,   103100B,   102600B,   102500B,
            1065000B,  1024000B,  1064000B,  1026000B,  1066000B,  1022000B,
            1023000B,  1027000B,  1021000B,  1030200B,  1011000B,  1000400B,
            1010400B,  1001000B,  1011000B,  1031010B,  1022010B,  1023010B,
            1021010B,     000B,     100B,     110B,     120B,     130B,
              140B,     170B,     150B,     160B,     500B,     510B,
              520B,     530B,     540B,     570B,     550B,     560B,
                4B,       1B,     400B,     240B,     300B,     340B,
             2020B,    2004B,    2102B,    2040B,    2200B,    2300B,
             2001B,     640B,     700B,     740B,     600B,     604B,
             6002B,  1002B,  1004B,  1042B,  1044B);

*SECOND OPCODE FOR SOME OF THE MULTIPLE-INSTRUCTIONS*
OPCODE2 =   (20B, 21B, 22B, 23B, 24B, 27B, 25B, 26B, 40220B, 40210B,
             40223, 40238, 40248, 40278, 40253, 40263, 21000B, 20100B,
             60100);

*TABLE USED TO INVERT CARD COLUMNS (USED FOR ABSOLUTE BINARY OUTPUT)*
INVERT   =   (0000B, 4000B, 2000B, 6000B, 1000B, 5000B, 3000B, 7000B,   *
              0400B, 4400B, 2400B, 6400B, 1400B, 5400B, 3400B, 7400B,   *
              0200B, 4200B, 2200B, 6200B, 1200B, 5200B, 3200B, 7200B,   *
              0600B, 4600B, 2600B, 6600B, 1600B, 5600B, 3600B, 7600B,   *
              0100B, 4100B, 2100B, 6100B, 1100B, 5100B, 3100B, 7100B,   *
              0500B, 4500B, 2500B, 6500B, 1500B, 5500B, 3500B, 7500B,   *
              0300B, 4300B, 2300B, 6300B, 1300B, 5300B, 3300B, 7300B,   *
              0700B, 4700B, 2700B, 6700B, 1700B, 5700B, 3700B, 7700B,   *
              0040B, 4040B, 2040B, 6040B, 1040B, 5040B, 3040B, 7040B,   *
              0440B, 4440B, 2440B, 6440B, 1440B, 5440B, 3440B, 7440B,   *
              0240B, 4240B, 2240B, 6240B, 1240B, 5240B, 3240B, 7240B,   *
              0640B, 4640B, 2640B, 6640B, 1640B, 5640B, 3640B, 7640B,   *
              0140B, 4140B, 2140B, 6140B, 1140B, 5140B, 3140B, 7140B,   *
              0540B, 4540B, 2540B, 6540B, 1540B, 5540B, 3540B, 7540B,   *
              0340B, 4340B, 2340B, 6340B, 1340B, 5340B, 3340B, 7340B,   *
              0740B, 4740B, 2740B, 6740B, 1740B, 5740B, 3740B, 7740B,   *
              0020B, 4020B, 2020B, 6020B, 1020B, 5020B, 3020B, 7020B,   *
              0420B, 4420B, 2420B, 6420B, 1420B, 5420B, 3420B, 7420B,   *
              0220B, 4220B, 2220B, 6220B, 1220B, 5220B, 3220B, 7220B,   *
              0620B, 4620B, 2620B, 6620B, 1620B, 5620B, 3620B, 7620B,   *
              0120B, 4120B, 2120B, 6120B, 1120B, 5120B, 3120B, 7120B,   *
              0520B, 4520B, 2520B, 6520B, 1520B, 5520B, 3520B, 7520B,   *
              0320B, 4320B, 2320B, 6320B, 1320B, 5320B, 3320B, 7320B,   *
              0720B, 4720B, 2720B, 6720B, 1720B, 5720B, 3720B, 7720B,   *
              0060B, 4060B, 2060B, 6060B, 1060B, 5060B, 3060B, 7060B,   *
              0460B, 4460B, 2460B, 6460B, 1460B, 5460B, 3460B, 7460B,   *
              0260B, 4260B, 2260B, 6260B, 1260B, 5260B, 3260B, 7260B,   *
              0660B, 4660B, 2660B, 6660B, 1660B, 5660B, 3660B, 7660B,   *
              0160B, 4160B, 2160B, 6160B, 1160B, 5160B, 3160B, 7160B,   *
              0560B, 4560B, 2560B, 6560B, 1560B, 5560B, 3560B, 7560B,   *
              0360B, 4360B, 2360B, 6360B, 1360B, 5360B, 3360B, 7360B,   *
              0760B, 4760B, 2760B, 6760B, 1760B, 5760B, 3760B, 7760B) ;*

*OTHER TABLES USED IN SOME OF THE PROCEDURES *

D        =   (20B, 40B, 10000B);

JT       =   (46, 54, 55, 56, 57, 63, 67, 73);
```

```
KT      =    (1, 2, 0, 1, 2, 1, 0, 2);

PT      =    (3, 2*5, 3*1, 4, 6, 7, 2, 2*3, 8, 3*1, 4, 6, 7);

COMP    =    (1,2,3,4,5,6,10,11,14,74,75,76,78,79,80,81);

FLPL    =    (E.FADE,E.FSBE,E.FMPE,E.FOVE);

*TABLE OF ASCII CODE FOR CHARACTERS ARRANGED WITH ASCENDING DISPLAY
 CODES*
ASCIICODE = (0008, 1018, 1028, 1038, 1048, 1058, 1068, 1078, 1108,
             1118, 1128, 1138, 1148, 1158, 1168, 1178, 1208, 1218,
             1228, 1238, 1248, 1258, 1268, 1278, 1308, 1318, 1328,
             0608, 0618, 0628, 0638, 0648, 0658, 0668, 0678, 0708,
             0718, 0728, 0738, 0748, 0758, 0768, 0778, 0038, 0148,
             0448, 0548, 0368, 0428, 1338, 1358, 0728, 0478, 0438,
             1008, 0468, 1368, 0418, 0748, 0768, 1348, 0778, 0458,
             0738);

*THE FOLLOWING TABLES CONTAIN CODED INFORMATION FOR TITLE PAGE,
HEADER CARD, ERROR CARD, ETC.*

HH = ( 4*EHH     HHE, 2*EHHHHHHHHHE, 6*EHH     HHE);
PP = ( EHHHHHHHE, EHHHHHHHE, 2*EHH     HHE, EHHHHHHHHE, EHHHHHHHE,
       6*EHHE);

DIG = (2*E   HHHHHHE, 8*E HH     HHE, 2*E   HHHHHHE,
       2*E     HHE, 2*E     HHHE, 2*E   HHHHHE, 4*E     HHE, 2*E   HHHHHHE,
       2*E HHHHHHHE, 2*E HH     HHE, 2*E     HHE, 2*E     HHE, 2*E HHE,
       2*EHHHHHHHHE, 2*E   HHHHHHE, 2*E HH     HHE, 2*E     HHHE,
       2*E         HHE, 2*E HH     HHE, 2*E   HHHHHHE, 6*EHH     HHE,
       2*EHHHHHHHHHE, 4*E       HHE, 2*E HHHHHHHHE, 2*E HHE,
       2*E HHHHHHHE, 4*E         HHE, 2*E HHHHHHHE, 2*E  HHHE, 4*E HHE,
       2*E HHHHHHHE, 2*E HH     HHE, 2*E   HHHHHHE, 2*E  HHHHHHHE,
       2*E       HHE, 2*E     HHE, 2*E     HHE, 2*E   HHE,
       2*E  HHHHHHE, 2*E HH     HHE, 2*E   HHHHHHE, 4*E HH     HHE,
       4*E  HHHHHHE, 2*E HH     HHE, 2*E   HHHHHHE, 4*E       HHE,
       2*E     HHHE);

IMAGE = (600363636773641364138,  641364136773636360138,
         600360036113631367738,  677360136013600360038,
         600362136633643364538,  645365136513621360138,
         600362236633641364138,  651365136773626360138,
         600367436743604360438,  604367736773604360138,
         600367136713651365138,  651365136573646360138,
         600363636773645364538,  645364536473602360138,
         600364036413643336738,  654367036603640360138,
         600362636773651365138,  651365136773626360138,
         600363036753645364538,  645364536773636360138 );

HEDCARD = (3*600360036036036003600038, 600360036103677367738,
           610361036103610361038, 610367736773600367738,
           677365036503650365038, 650365036203620360038,
           5*600360036003600360038 );

ERRCARD = ( 3*0, 477147714511451145118, 451145114511400140018,
                 477147714501450145418, 456145214711421140318,
                 477147714501450145418, 456145214711421140018,
                 436143614411441144118, 441144114361436140018,
                 477147714501450145418, 456145214711421140018,
```

```
            3*0 );
*=============================================================*

PROCEDURE OUTCH(CH:CHAR);

*PRINT THE CHARACTER CH *

BEGIN
   OUTPUT^:=CH; PUT(OUTPUT)
END *OUTCH*;

*=============================================================*

PROCEDURE OUTINT(N:INTEGER);

*PRINT THE DECIMAL INTEGER N AND A BLANK*

VAR I,J : INTEGER; A : ARRAY [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 OUTCH(CHR(A[J]+27)); OUTCH(' ');
END *OUTINT*;

*=============================================================*

PROCEDURE OUTOCT(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;
   K,M,J,S,P : INTEGER;
BEGIN
   P:=21-NO;
   IF I=0 THEN FOR J:=1 TO NO DO WRITE('0') ELSE
  BEGIN
   F:=FALSE;
   M:=-7B; S:=3;
   FOR J:=1 TO 20 DO
        BEGIN
             K:=-I; APPEND(K,S,M);
             IF (K<>0)*(J=P) THEN F:=TRUE;
           IF F THEN OUTCH(CHR(27-K)); S:=S+3
        END;
  END; OUTCH(' ');
END * OUTOCT *;

*=============================================================*

PROCEDURE NEWPAGE;
*WRITE ON A NEW PAGE*

BEGIN
   PG:=PG+1; WRITE(EOL,'1',' PAGE ',PG:5,'          ');
   LI:=2
END *NEWPAGE*;

*=============================================================*
```

```
PROCEDURE READARRAY(VAR A:ARRCHAR90);

*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:ARRCHAR90);

* 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(' ');
   LI:=LI+1; IF LI>LP THEN BEGIN
                             NEWPAGE;
                             PRINTARRAY(HDNG); WRITE(EOL,EOF,EOL,' ')  *
                           END
END *PRINTARRAY*;

*=============================================================*
PROCEDURE COPYARRAY(A:ARRCHAR90; VAR B:ARRCHAR90);

*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(' ')
END *SPACE*;

*=============================================================*

PROCEDURE MULTIPRINT(CH:CHAR;N:INTEGER);

*PRINT THE CHARACTER CH REPEATEDLY N TIMES*

VAR I:INTEGER;
BEGIN
   FOR I:=1 TO N DO OUTCH(CH)
END *MULTIPRINT*;
```

```
*=============================================================================*

PROCEDURE TITLE;

*PRINT PROGRAM HEADING ON THE TOP OF THE NEXT PAGE*

BEGIN

   NEWPAGE; PRINTARRAY(HDNG); WRITE(EOL,ΞCΞ,EOL,Ξ Ξ)
END *TITLE*;

*=============================================================================*

PROCEDURE LINES(N:INTEGER);

*PRINT N BLANK LINES*

VAR I:INTEGER;
BEGIN
   LI:=LI+N;
   IF LI>LP THEN TITLE ELSE FOR I:=1 TO N DO WRITE(EOL,Ξ Ξ)
END *LINES*;
*=============================================================================*

PROCEDURE PRINTTITLE;
BEGIN
   FIRSTERR:=FALSE;
   WRITE(ΞERRORS DETECTED IN PASS 1Ξ); LINES(1);
   MULTIPRINT(Ξ*Ξ,25); LINES(2);
 WRITE(Ξ(ERROR MESSAGES ARE LISTED BELOW THE ERRONOUS LINES)Ξ); LINES(4)
END *PRINTTITLE*;
*=============================================================================*

PROCEDURE ERROR(N:INTEGER);

*STORE ERROR NO.; UPDATE ERROR COUNTERS:
      ER=CUMULATIVE ERROR COUNT FOR THE CURRENT PASS,
      NER=ERROR COUNT FOR THE CURRENT STATEMENT,
      ERRSTORE=STORES THE ERROR NO(S). FOR THE CURRENT STATEMENT,
      ERRPAGE=POWERSET THAT STORES THE PAGE NUMBER*

BEGIN
   ERRFLAG:=TRUE;
   ER:=ER+1; IF NER<MXER THEN BEGIN NER:=NER+1; ERRSTORE[NER]:=N END;
   IF ~(N IN NONFATAL) THEN FATAL:=TRUE;
   IF PG<58 THEN BEGIN ER:=PG; ERRPAGE:=ERRPAGE + [PG] END
END *ERROR*;

*=============================================================================*

PROCEDURE ERRMESSAGE(N:INTEGER);

*PRINT MESSAGE CORRESPONDING TO ERROR NO. N*

PROCEDURE NL;   BEGIN LINES(1); SPACE(19) END;

BEGIN
  WRITE(Ξ      ***** ERROR: Ξ);
   IF N<30 THEN
   CASE N OF
    0 :WRITE(ΞEOF ENCOUNTERED, END ASSUMEDΞ);
```

```
 1 !WRITE(=SYMBOL TOO LONG, TRUNCATED TO 9 CHARACTERS=);
 2 !WRITE(=ILLEGAL CHARACTER IN SYMBOL=);
 3 !WRITE(=ILLEGAL OPERATION=);
 4 !WRITE(=UNDECLARED VARIABLE=);
 5 !WRITE(=ILLEGAL CHARACTER IN OPERAND FIELD=);
 6 !WRITE(=ILLEGAL OPERATION IN AN ABSOLUTE PROGRAM=);
 7 !WRITE(=ILLEGAL CHARACTER IN AN OCTAL CONSTANT=);
 8 !WRITE(=CONSTANT GREATER THAN 177777=);
 9 !WRITE(=ERROR IN PSEUDO-OP ASC; N<1 OR N>28=);
10 ! WRITE(=NON-BLANK OPERAND FIELD EXPECTED=);
11 !WRITE(=DOUBLY DEFINED SYMBOL=);
12 !WRITE(=TOO MANY DISTINCT LITERALS; MAX=,MLT+3);
13 !WRITE(=ILLEGAL @P IN MULTIPLE INSTRUCTION=);
14 !WRITE(=MIXING OF A, B TYPE INSTRUCTIONS IS ILLEGAL=);
15 !WRITE(=ILLEGAL MIXING OF SRG AND ASG INSTRUCTIONS=);
16 !WRITE(=ILLEGAL SEQUENCE OF OPS IN A MULTIPLE INSTRUCTION=);
17 !WRITE(=INVALID LITERAL=);
18 !WRITE(=ASMB,  EXPECTED IN COLS. 1 - 5 OF THE CONTROL CARD=);
19 !WRITE(=ILLEGAL PARAMETER IN CONTROL STATEMENT=);
20 !WRITE(=ILLEGAL CHARACTER IN CONSTANT=);
21 !WRITE(=CONTROL CARD ERROR; NEITHER A OR R OR BOTH SPECIFIED=);
22 !WRITE(=CONTROL CARD ERROR; NO OUTPUT PARAMETER SPECIFIED=);
23 !WRITE(=SYMBOL TABLE OVERFLOW; MAX=,S7+4);
24 !WRITE(=ABOVE ENT SYMBOL APPEARS IN A COM OR AN EXT STATEMENT=);
25 !WRITE(=ABOVE ENT SYMBOL IS NOT A LABEL OF ANY INSTRUCTION=);
27 !WRITE(=TOO MANY ENT SYMBOLS; MAX=,MNT+3);
28 !WRITE(=LABEL ILLEGAL IN AN INSTRUCTION FOLLOWING REP PSEUDO=);
29 ! BEGIN WRITE(=CHARACTER FOLLOWING COMMA IS INCOMPATIBLE WITH =);
        NL;WRITE(=THE INSTRUCTION USED=)
    END;
END ELSE
CASE N OF
30 !WRITE(=PROGRAM RELOCATABLE OPERAND EXPEXTED=);
31 !WRITE(=ABSOLUTE OPERAND EXPECTED=);
32 !WRITE(=SIGN MISSING BETWEEN TWO TERMS IN OPERAND=);
33 ! BEGIN WRITE(=OPERAND IS A MIXTURE OF PROGRAM, BASE PAGE, =);
        NL; WRITE(=EXTERNAL AND COMMON RELOCATABLE TERMS=) END;
34 !WRITE(=EXTERNAL SYMBOL SHOULD APPEAR ALONE IN OPERAND=);
35 !WRITE(=ABSOLUTE OPERAND IN A RELOCATABLE PROGRAM EXCEEDS 1008=);
36 !WRITE(=OPERAND IS NEGATIVE FOR AN INSTRUCTION OTHER THAN ABS=);
37 !WRITE(=EXPRESSION IN AN L-TYPE LITERAL MUST BE ABSOLUTE=);
38 !WRITE(=LITERAL INCOMPATIBLE WITH THE INSTRUCTION USED=);
39 ! IF ABSPROG THEN
    BEGIN
   WRITE(=FIRST OPCODE IN AN ABS PROGRAM DOES NOT SUCCEED AN ORG =);
  NL;WRITE(=STATEMENT. PROGRAM ASSEMBLED STARTING AT ADDRESS 2000B=)
    END ELSE
    BEGIN
   WRITE(=FIRST OPCODE IN A RELOCATABLE PROGRAM DOES NOT SUCCEED =);
     NL; WRITE(=A NAM STATEMENT=);
    END;
40 !WRITE(=OPERAND EXCEEDS 17778 FOR A MEMORY REF. INSTRUCTION=);
41 !WRITE(=OPERAND EXCEEDS 778 FOR AN I/O GROUP INSTRUCTION=);
42 !WRITE(=OPERAND EXCEEDS 178 FOR A REGISTER REF. INSTRUCTION=);
43 !WRITE(=OPERAND EXCEEDS 777778=);
44 !WRITE(=OUT OF RANGE REAL CONSTANT=);
45 ! BEGIN WRITE(=AN IFN OR AN IFZ FOLLOWS EITHER AN IFN OR AN IFZ =);
        NL; WRITE(=WITHOUT AN INTERVENING XIF. THE SECOND PSEUDO =);
        NL; WRITE(=INSTRUCTION IS IGNORED=)
    END;
46 !WRITE(=HPASMB.  EXPECTED IN COLS. 1 - 7 OF THE USER'S CARD=);
```

```
   47: BEGIN WRITE('OPERAND ADDRESS IS NOT ON THE SAME PAGE OF MEMORY ')
       : NE: WRITE('AS THE INSTRUCTION. (ABS PROGRAM)') END;
   48: WRITE('NEGATIVE OR DOUBLE RELOCATION IN OPERAND');
   END;
   LINES(1)
END *ERRORMESSAGE*;

{=============================================================================}

PROCEDURE PRINTERR;

VAR I:INTEGER;
*PRINT ERRORS IN THE LAST STATEMENT*

BEGIN
   FOR I:=1 TO NER DO ERRMESSAGE(ERRSTORE[I]); LINES(1);
   NER:=0;
END *PRINTERR*;

{=============================================================================}

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+1; 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);
*INCREMENT LOCATION COUNTER BY N.
 IF PAGEFLAG=TRUE -- CURRENT PAGE LOADING, LOCATION COUNTER=LC,
          =FALSE -- BASE PAGE LOADING, LOCATION COUNTER=BPC*
BEGIN
   IF PAGEFLAG THEN LC:=LC+N ELSE BPC:=BPC+N
END *OF INCRCOUNT* ;
*=============================================================*
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=' ')+(CH=',')
END *OF BLANKORCOMMA*;
*=============================================================*
FUNCTION BLANKOREOL(CH:CHAR):BOOLEAN;
* IS CH A BALNK OR AN EOL CHAR *
BEGIN
   BLANKOREOL:=(CH=' ')+(CH=EOL)
END *BLANKOREOL*;
*=============================================================*
FUNCTION ENDCHAR(CH:CHAR):BOOLEAN;
*IS CH A BLANK OR COMMA OR EOL*
BEGIN
   ENDCHAR:=(CH=' ')+(CH=',')+(CH=EOL)
END *ENDCHAR*;
*=============================================================*
PROCEDURE SKIPREST;
*SKIP ALL THE CHARACTERS TILL A BLANK OR COMMA*
BEGIN
```

```
   ERROR(5);
   REPEAT
      NEXTCHAR(CH)
   UNTIL BLANKORCOMMA(CH);
END (*OF SKIPREST*);

{=============================================================================*}

PROCEDURE LOCTYP(VAR LOC:INTEGER; VAR TYP:CHAR);

(*PUT THE VALUE OF THE CURRENT LOC. COUNTER IN LOC AND THE PAGE INFOR-
 MATION IN TYP. (TYP=R FOR CURRENT PAGE, TYP=B FOR BASE PAGE)*)

BEGIN
   IF PAGEFLAG THEN BEGIN LOC:=LC; TYP:='R' END
               ELSE BEGIN LOC:=BPC; TYP:='B' END;
   IF NOT SCAN1 THEN LOC:=LC
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; I: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 N>=0 THEN POWER:=Z ELSE POWER:=1/Z
END (*POWER*);

{=============================================================================*}

PROCEDURE SQUEEZE(VAR B:ARRCHAR90; VAR A:ARRALFA9);

(*PACKS CHARACTER ARRAY B[90] INTO AN ALFA ARRAY A[9] SO THAT 10 CHAR-
 ACTERS ARE PACKED TO AN ALFA WORD*)

VAR I,J:INTEGER;
BEGIN J:=0;
   REPEAT J:=J+1 UNTIL B[J]=EOL;
   FOR I:=J TO 89 DO B[I]:=' '; B[90]:=EOL;
   J:=1;
   FOR I:=1 TO 9 DO BEGIN PACK(B,J,A[I]); J:=J+10 END
END (*SQUEEZE*);

{=============================================================================*}

PROCEDURE EXPAND(VAR A:ARRALFA9; VAR B:ARRCHAR90);

(*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#0 THEN *ERRORS IN PREVIOUS LINE. PRINT LINE AND ERROR NOS.*
   BEGIN
      IF FIRSTERR THEN PRINTTITLE;
      WRITE(LN,'  ');          PRINTARRAY(INTEXT); PRINTERR
   END;

   READARRAY(INTEXT);
   IF  ENDINPUT  THEN BEGIN
                         ERROR(1);
                         UNPACK('  END  ',INTEXT,1);
                         INTEXT(11):=EOL; SQUEEZE(INTEXT,S*.LINE)
                      END
   ELSE BEGIN
          LN:=LN*1; S*.LN:=LN; SQUEEZE(INTEXT,S*.LINE);
          IF INTEXT(1)='*' THEN *COMMENT CARD; READ ANOTHER LINE*
             BEGIN S*.OPFLD:=0; S*.LOCFLD:=LC; PUT(S); GOTO 1 END;
        END
END *READLINE*;
*=============================================================================*

PROCEDURE WRITELINE;

*WRITE THE CURRENT LINE, PRINT ERRORS, IF ANY*

   PROCEDURE WRITEOUT;
   VAR I:INTEGER;
   BEGIN
      FOR I:=1 TO 9 DO WRITE(S*.LINE[I]); OUTCH(' ');
      LI:=LI*1; IF LI>LP THEN TITLE
   END *PRINTOUT*;

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 *WRITELINE*;
*=============================================================================*

PROCEDURE STORELINE;

*STORE INFORMATION ABOUT THE CURRENT STATEMENT ON A FILE OF RECORDS,
 FOR LATER USE IN PASS TWO*

BEGIN
   IF PAGEFLAG THEN S*.LOCFLD:=LC ELSE S*.LOCFLD:=BPC;
   S*.OPFLD:=OPN
END *STORELINE*;
```

```
{=============================================================================}
PROCEDURE COPYCARD;
{ WRITE LINE WITH PROPER SPACING }
BEGIN
   SPACE(20); WRITELINE
END {COPYCARD};
{=============================================================================}
PROCEDURE STOREWORD(L,W: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 PUTFILE(VAR F:FILEOUTCODE);
   BEGIN
      WITH F^ DO BEGIN LOC:=L; WORD:=W; TYP:=T; MREF:=MR END; PUT(F)
   END {PUTFILE};
BEGIN
   IF BINOUTPUT THEN
   IF PAGEFLAG THEN PUTFILE(R) ELSE PUTFILE(B);
   IF LSTFLAG OR (NER<>0) THEN
   IF EXTRALINE THEN BEGIN
                     IF NOT SUPFLAG THEN BEGIN
                                           SPACE(2); OUTOCT(L);
                                           OUTOCT(W); OUTCH(T); LINES(1)
                                         END;
                     IF NER<>0 THEN PRINTERR
                   END
               ELSE BEGIN
                     N:=W;
                     IF MR>=0 THEN {MEMORY REF. INSTR.}
                                BEGIN K:=MR MOD 2000B; APPEND(N,0,K) END;
                     OUTOCT(L); OUTOCT(N); OUTCH(T); SPACE(5);
                     WRITELINE
                   END
END {STOREWORD};
{=============================================================================}
PROCEDURE CHECKSIGN(VAR S:BOOLEAN);
{CHECK SIGN OF A NUMBER AND ADVANCE TO NEXT CHARACTER}
BEGIN
   CURRCHAR(CH);
   S:=TRUE;
   IF CH='+' THEN NEXTCHAR(CH) ELSE
   IF CH='-' THEN BEGIN S:=FALSE; NEXTCHAR(CH) END
END {OF CHECKSIGN};
{=============================================================================}
```

```
PROCEDURE GETOCT(VAR N:INTEGER);
*EVALUATE AN OCTAL INTEGER CONSTANT, CHECK FOR ERRORS*

VAR S:BOOLEAN;
BEGIN
   SKIPBLANKS; CHECKSIGN(S); N:=0;
   WHILE (ORD(CH)>=ORD("0")) AND (CH<="7") DO BEGIN N:=N*8+ORD(CH)-27; NEXTCHAR(CH) END;*
   IF DIGIT(CH) OR (CH="8") OR (CH="9") THEN ERROR(7);
   IF N>1777778 THEN BEGIN N:=0; ERROR(8) END;
   IF S THEN N:=2000000B-N  *2,S COMPLEMENT*;
END *OF GETOCT*;

*=============================================================================*

PROCEDURE GETNUM(VAR N:INTEGER);

*EVALUATE AN INTEGER. TREAT IT AS OCTAL IF FOLLOWED BY B *

VAR
   S,F:BOOLEAN; I,J,BASE : INTEGER;
   A:ARRAY[1..6] OF CHAR;
BEGIN
   F:=FALSE; I:=0; N:=0;
   SKIPBLANKS; CHECKSIGN(S);
   WHILE DIGIT(CH) DO
   BEGIN
      IF I<6 THEN BEGIN I:=I+1; A[I]:=CH END ELSE F:=TRUE;
      NEXTCHAR(CH)
   END;
   IF F THEN GOTO 1;
   IF CH="B" THEN BEGIN BASE:=8; NEXTCHAR(CH) END ELSE BASE:=10;
   FOR J:=1 TO I DO N:=N*BASE+ORD(A[J])-27;
   IF N GT 1777779 THEN
1: BEGIN ERROR(8); N:=0;  GOTO 2 END;
   IF S THEN N:=2000000B-N  *2,S COMPLEMENT*;
2:
END *OF GETNUM*;

*=============================================================================*
PROCEDURE GETCONST(VAR KIND:INTEGER; VAR X:REAL; VAR N:INTEGER);

*EVALUATE THE NEXT CONSTANT, REAL OR INTEGER. THE RESULT IS CODED AS
 FOLLOWS *
    * KIND=0 --- ERROR IN CONSTANT, RETURN X=0.0, N=0,
      KIND=1 --- INTEGER CONSTANT N, X=0.0,
      KIND=2 --- REAL CONSTANT, X=MANTISSA, N=EXPONENT TO BASE 10 *

VAR S:BOOLEAN; F,Y:REAL;
BEGIN
   KIND:=2; X:=0.0;
   SKIPBLANKS; CHECKSIGN(S);
   IF DIGIT(CH) THEN BEGIN
                     GETNUM(N); IF ~S THEN N:=-N; X:=N;
                     IF CH="E" THEN GOTO 3
                     ELSE IF CH="." THEN
                                    BEGIN
                                     NEXTCHAR(CH);
                                     IF CH="E" THEN GOTO 3
                                     ELSE IF DIGIT(CH) THEN GOTO 4  *
                                     ELSE IF ENDCHAR(CH) THEN GOTO 5*
```

```
                                                ELSE GOTO 2
                                              END
                                ELSE IF ENDCHAR(CH) THEN
                                                BEGIN KIND:=1; X:=C.0; GOTO 6 END *
                                ELSE GOTO 2
                              END
    ELSE IF CH="." THEN BEGIN
                              NEXTCHAR(CH);
                              IF DIGIT(CH) THEN GOTO 4
                              ELSE GOTO 2
                            END
    ELSE
2:  *ILLEGAL CHARACTER*
    KIND:=0; SKIPREST; X:=C.0; N:=0; GOTO 6;

3:  * E *
    NEXTCHAR(CH);
    CHECKSIGN(S);
    IF DIGIT(CH) THEN BEGIN
                        GETNUM(N); IF ~S THEN N:=-N;
                        IF ENDCHAR(CH) THEN GOTO 6 ELSE GOTO 2
                      END
    ELSE GOTO 2;

4:  * DIGIT AFTER PERIOD *
    F:=1.0; Y:=0.0;
    WHILE DIGIT(CH) DO BEGIN
                         F:=F/10;
                         Y:=Y+F*(ORD(CH)-27);
                         NEXTCHAR(CH)
                       END;
    IF ~S THEN Y:=-Y; X:=X+Y;
    IF CH="E" THEN GOTO 3
    ELSE IF ENDCHAR(CH) THEN GOTO 5
    ELSE GOTO 2;

5:  N:=0;
6:  * BLANK OR COMMA. CONSTANT EVALUATED. CHECK RANGE *
    IF X<>0 THEN BEGIN
                   Y:=ABS(X*POWER(10.0,N));
                   IF (Y<MINREAL)∨(Y>MAXREAL) THEN ERROR(44)
                 END

END *GETCONST*;

{=============================================================================*

PROCEDURE CONVERT(P:PRECISION; VAR X:REAL; VAR N,W1,W2,W3:INTEGER);

    * CONVERTS REAL CONSTANT X=N (=X*10**N) INTO HP WORDS AS FOLLOWS:
      IF P=SINGLE THEN SINGLE PRECISION, X=N IS RE-STORED IN W1,W2
      IF P=DOUBLE THEN EXTENDED PRECISION, X=N IS RE-STORED IN W1,W2,W3*

VAR
  M: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; X:=-X END
  ELSE S:=TRUE;
```

```
    X1=X*POWER(1.25,N);
    N1=3*N;  *EXPONENT TO BASE 2*
    * NORMALIZE FRACTION *
    WHILE X>1.0   DO BEGIN X1=X/8; N1=N+3 END;
    WHILE X<0.125 DO BEGIN X1=8*X; N1=N-3 END;
    WHILE X<0.5   DO BEGIN X1=2*X; N1=N-1 END;
    X1=X+EPSILON; IF X>0.999999 THEN BEGIN X1=0.5; N1=N+1 END;
    * EXPONENT *
    N1=2*N;
    IF N<0 THEN N1=N+4018;      * 2,S COMPLEMENT IF EXPONENT IS NEGATIVE *

    IF P=DOUBLE THEN *EXTENDED PRECISION* GOTO 2;

    * ORDINARY PRECISION CONSTANT *
    M1=TRUNC(X*4000000(8);
    IF ~S THEN M1:=-M+100000000B;     * 2,S COMPLEMENT IF NEGATIVE*

    W11=M DIV 4000B; W21=M MOD 4000B; APPEND(W2,8,N);

    * SINGLE PRECISION CONSTANT COMPLETED *
    W31=0;
    GOTO 3;

2:  * EXTENDED PRECISION CONSTANT TO BE STORED IN THREE H.P. WORDS *
    M1:= TRUNC(X*1000000000000(8);

    IF ~S THEN M1:=-M+200000000000000B;
    *PREPARE FIRST WORD*
    W11=M DIV 1000000000(8;
    M1:=M MOD 1000000000B;
    *PREPARE SECOND WORD*
    W21=M DIV 4000B;
    *START THIRD WORD*
    W31=M MOD 4000B;
    APPEND(W3,8,N);

    * EXTENDED PRECISION CONSTANT COMPLETED *

3:
END *CONVERT*;

*=============================================================================*

PROCEDURE STOREINT(VAR N:INTEGER);

*STORE INTEGER AND THE LOC. COUNTER FOR PASS TWO*

BEGIN
    IF N<0 THEN N:=20000(8+N *2,S COMPLEMENT*;
    STOREWORD(LC,N,= =,MR); LC:=LC+1; EXTRALINE:=TRUE;
END *OF STOREINT*;

*=============================================================================*

PROCEDURE STOREREAL(PRECISION:PRECISION; VAR X:REAL; VAR N:INTEGER);

* STORE A REAL CONSTANT IN TWO WORDS IF SINGLE PRECISION, AND THREE
WORDS IF EXTENDED PRECISION *

VAR W1,W2,W3 : INTEGER;
```

```
BEGIN
   CONVERT(P,X,N,W1,W2,W3);
   STOREWORD(LC,W1,Ξ Ξ,MR); EXTRALINE:=TRUE;
   STOREWORD(LC+1,W2,Ξ Ξ,MR);
   IF PESINGLE THEN LC:=LC+2
               ELSE BEGIN STOREWORD(LC+2,W3,Ξ Ξ,MR); LC:=LC+3 END;
END *OF STOREREAL*;

*=============================================================================*

PROCEDURE GETNAME(VAR A:ALFA);

*GET THE NEXT SYMBOL. 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 I:INTEGER; F:BOOLEAN; C:ARRCHAR10;
BEGIN
   SKIPBLANKS; F:=FALSE;
   IF CH IN ALPHA THEN C[1]:=CH
                  ELSE BEGIN
                          ERROR(2); A:=EMPTY;
                          REPEAT NEXTCHAR(CH) UNTIL (CH>Ξ9Ξ)^(CH≠Ξ.Ξ);*
                          GOTO 1
                       END;
   NEXTCHAR(CH); I:=1;
   WHILE (CH IN ALPHA)^(CH IN DIGITS) DO
   BEGIN
      IF I<5 THEN BEGIN I:=I+1; C[I]:=CH END ELSE F:=TRUE;
      NEXTCHAR(CH)
   END;
   IF F THEN ERROR(1);
   REPEAT I:=I+1; C[I]:=Ξ Ξ UNTIL I=10;
   PACK(C,1,A);
1:
END * OF GETNAME *;

*=============================================================================*

PROCEDURE SKIPTOEND;

*SKIP ALL THE SOURCE STATEMNETS TILL THE END PSUEDO. USED TO SKIP A
 PROGRAM WITH ERRORS IN CONTROL STATEMENT*

VAR A:ALFA;
BEGIN
   REPEAT
      READARRAY(INTEXT);
      IF EOF(INPUT) THEN BEGIN ERROR(0); GOTO 1 END;
      COL:=1; SKIPTOBLANK; GETNAME(A)
   UNTIL A=ΞENDΞ;
1:
END *SKIPTOEND*;

*=============================================================================*

PROCEDURE ENTER(ID:ALFA; L:INTEGER; T:CHAR);

*ENTER THE SYMBOL, THE LOC. COUNTER AND THE RELOCATIBILITY OF THE SYMBOL*
 IN THE SYMBOL TABLE USING HASH-CODING. A CHECK IS MADE FOR DUPLICATE
 ENTRIES. THE RELOCATABILITY OF THE SYMBOL IS CODED AS FOLLOWS:
```

```
  P=PROG. RELOCATABLE, B=BASE PAGE RELOCATABLE, C=COMMON, X=EXTERNAL,    *
  BLANK=ABSOLUTE *

VAR
   F : BOOLEAN;
   H,D : INDEXS;

BEGIN
   F:=FALSE; D:=1;
   H:=((ABS(ORD(ID))) DIV 4096) MOD SZ;
   REPEAT *SEARCH TABLE AND STORE IF NOT ALREADY PRESENT*
        IF SYMTAB[H].NAME=ID THEN *FOUND* BEGIN ERROR(11); F:=TRUE END    *
                     ELSE IF SYMTAB[H].NAME=EMPTY THEN
                          BEGIN *NEW ENTRY*
                               F:=TRUE;
                               SN:=SN+1; SYMPTR[SN]:=H;
                               WITH SYMTAB[H] DO
                                    BEGIN NAME:=ID; LOC:=L; TYP:=T END    *
                          END ELSE
                          BEGIN *COLLISION*
                               H:=H+D; D:=D+1;
                               IF D=SZ THEN BEGIN
                                                F:=TRUE; ERROR(23)
                                            END;
                               IF H≥SZ THEN H:=H MOD SZ;
                          END;
   UNTIL F;
END * OF ENTER *;

{=============================================================================}

PROCEDURE SEARCH(ID:ALFA; VAR LOC:INTEGER; VAR TYP:CHAR);

*CHECK IF THE GIVEN SYMBOL IS PRESENT IN THE SYMBOL TABLE. IF PRESENT     *
 THEN EXTRACT ITS LOCATION AND RELOCATIBILITY AND TURN THE BOOLEAN
 VARIABLE PRESENT=TRUE, OTHERWISE SIMPLY TURN PRESENT=FALSE*

VAR
   F : BOOLEAN;
   H,D : INDEXS;

BEGIN
   F:=FALSE; D:=1;
   H:=((ABS(ORD(ID))) 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:=E    *
                          END ELSE
                          BEGIN *COLLISION*
                               H:=H+D; D:=D+1;
                               IF D=SZ THEN BEGIN *TABLE QUASI FULL *
                                                F:=TRUE; PRESENT:=FALSE   *
                                            END;
                               IF H≥SZ THEN H:=H MOD SZ;
```

```
                    END;
  UNTIL F;
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
  COL:=1;
  IF INTEXT(1) = ' ' THEN LBL:=EMPTY
  ELSE BEGIN
         GETNAME(LBL);
         IF CH<>' ' THEN BEGIN ERROR(2); SKIPTOBLANK 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
                       LOCTYP(LOC,TYP);
                       ENTER(LBL,LOC,TYP)
                     END
END *OF ENTERLABEL*;
*=============================================================================*

PROCEDURE GETOP(VAR OPN:INTEGER);

*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:INTEGER);  *LOCAL PROCEDURE*

  VAR I,J,K:INTEGER; OP:ALFA; C:ARRCHAR10;
  BEGIN *EXTRACT OP FIELD *
    I:=0;
    REPEAT
      I:=I+1; C[I]:=CH; NEXTCHAR(CH)
    UNTIL ENDCHAR(CH);
    REPEAT I:=I+1; C[I]:=' ' UNTIL I>10;
    PACK(C,1,OP);

    *CHECK OP IN OPTABLE*
     I:=1; J:=NI;
     REPEAT
           K:=(I+J) DIV 2;          * BINARY SEARCH *
           IF OPTABLE[K] < OP THEN I:=K+1;
           IF OPTABLE[K] > OP THEN J:=K-1
```

```
      UNTIL I>J;

      *IF VALID OP THEN GET ITS SERIAL NUMBER ELSE SERIAL NO = 0*
      IF OPTABLE[K]=OP THEN N:=OPNUM[K] ELSE N:=0
   END *CHECKOP*----------------;

PROCEDURE DUMMYLINE(OPN:INTEGER);

BEGIN
   S*.LN:=LN; S*.LOCFLD:=LC; S*.OPFLD:=-OPN;
   SQUEEZE(INTXT,S*.LINE);
END *DUMMYLINE*-------------------------------------------------------*;

BEGIN *MAIN PROCEDURE*
   CHECKOP(OPN);
   IF (CH=",")^(OPN<>0) THEN *MULTIPLE INSTRUCTION*
   BEGIN
      OPN:=-OPN; STORELINE; PUT(S);
      WHILE CH="," DO
      BEGIN
         NEXTCHAR(CH); CHECKOP(OPN);
         IF OPN=0 THEN OPN:=1;
         DUMMYLINE(OPN); PUT(S);
      END;
   DUMMYLINE(0); 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; I:INTEGER;
BEGIN
   NEXTCHAR(C1); NEXTCHAR(C2);
   N:=ASCIICODE[C1]; I:=ASCIICODE[C2];
   APPEND(N,8,I) *SHIFT N 8 BITS TO LEFT, TAKE LOGICAL OR WITH I*
END *GETASC*;

{=============================================================================*}

PROCEDURE SCANOPERAND(VAR NUM:INTEGER; VAR TYP: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,W1,W2,I);
        FOR I:=1 TO MLT-1 DO
        BEGIN
           IF (LITTAB[I,1]=W1)&(LITTAB[I+1,1]=W2) THEN {LITERAL PRESENT}
              BEGIN NUM:=LITTAB[I,2]; GOTO 1 END
           ELSE IF LITTAB[I,1]=NOLIT THEN {NEW LITERAL}
              BEGIN
                 LITTAB[I,1]:=W1; LITTAB[I,2]:=ENDLC;
                 LITTAB[I+1,1]:=W2; LITTAB[I+1,2]:=ENDLC+1;
                 NUM:=ENDLC; ENDLC:=ENDLC+2; GOTO 1
              END
        END; ERROR(12);
     END ELSE ERROR(17);
1:
END {FLITERAL};
```

```
PROCEDURE LITERAL(VAR NUM:INTEGER);

{ROUTINE TO CHECK AND STORE ALL LITERALS}

VAR I,N:INTEGER; T:CHAR;

   FUNCTION COMPATIBLE(CH:CHAR; OPN:INTEGER):BOOLEAN;

   {CHECK IF LITERAL IS COMPATIBLE WITH THE INSTRUCTION USED}

   VAR I,N:INTEGER; B:BOOLEAN;
   BEGIN
      B:=FALSE;
      IF CH='F' THEN BEGIN I:=0; N:=11 END ELSE BEGIN I:=11; N:=16 END;
      REPEAT
         I:=I+1;
         IF OPN=COMP[I] THEN B:=TRUE;
      UNTIL B OR (I=N);
   COMPATIBLE:=B
   END {COMPATIBLE};
```

```
BEGIN
   NUM:=0; NEXTCHAR(CH);
   IF NOT (CH IN LITCHARS) THEN ERROR(17)
   ELSE IF COMPATIBLE(CH,OPN) THEN
   BEGIN {EXTRACT LITERAL}
      CASE CH OF
        'A': GETASC(N);
        'B': BEGIN NEXTCHAR(CH); GETOCT(N) END;
        'D': BEGIN NEXTCHAR(CH); GETNUM(N) END;
        'F': BEGIN FLITERAL(NUM); GOTO 1 END;
        'L': BEGIN
                NEXTCHAR(CH); SCANOPERAND(N,T); IF T<>'=' THEN ERROR(37)
             END;
      END;

      {CHECK OLD LITERALS, STORE IF LITERAL NOT STORED BEFORE}
      FOR I:=1 TO MLT DO
```

```
      IF LITTAB[I,1]=N THEN BEGIN *LITERAL PRESENT*
                               NUM:=LITTAB[I,2]; GOTO 1
                            END
      ELSE IF LITTAB[I,1]=NOLIT THEN *NEW LITERAL*
                            BEGIN
                               LITTAB[I,1]:=N; LITTAB[I,2]:=ENDLC;
                               NUM:=ENDLC; ENDLC:=ENDLC+1; GOTO 1
                            END; ERROR(12);
  END ELSE ERROR(38);
1: CH:=EOL;
END *LITERAL*;
{-------------------------------------------------------------------}

  PROCEDURE ADNUM;
  BEGIN
    IF SIGN=1 THEN NUM:=NUM+N ELSE NUM:=NUM-N
  END *ADNUM*;
{-------------------------------------------------------------------}

  PROCEDURE NEXTTERM(VAR SIGN,N:INTEGER; VAR T:CHAR);

*EVALUATE THE VALUE (OR ADDRESS) OF THE NEXT TERM IN THE OPERAND*

  VAR A:ALFA;
  BEGIN
    SIGN:=1; T:=' ';
    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,N,T);
                            IF ~PRESENT THEN ERROR(4)
                         END
    ELSE IF DIGIT(CH) THEN BEGIN GETNUM(N); T:='#' END
    ELSE IF CH='*' THEN BEGIN LOCTYP(N,T); NEXTCHAR(CH) END
    ELSE IF CH='=' THEN BEGIN T:='R'; LITERAL(N) END
    ELSE BEGIN ERROR(5); NEXTCHAR(CH) END;
    FIRST:=FALSE
  END *NEXTTERM*;
{-------------------------------------------------------------------}

*MAIN PROCEDURE BODY -- SCANOPERAND*

BEGIN
  ERRFLAG:=FALSE; SKIPBLANKS;
  IF CH=EOL THEN *NO OPERAND FOUND* BEGIN ERROR(10); GOTO 1 END;
  NUM:=0; TYP:=' '; FIRST:=TRUE; INDIRECT:=FALSE;
  IF ABSPROG THEN *ABSOLUTE PROGRAM*
    REPEAT *EAVALUATE EACH TERM OF OPERAND EXPRESSION, KEEP ADDING*
      NEXTTERM(SIGN,N,T); ADNUM
    UNTIL ENDCHAR(CH)
  ELSE *RELOCATABLE PROGRAM*
  BEGIN
    NET:=0; COUNT:=0;
    REPEAT
      NEXTTERM(SIGN,N,T); COUNT:=COUNT+1;
      IF T=' ' THEN ADNUM
      ELSE IF TYP=' ' THEN BEGIN TYP:=T; ADNUM; NET:=NET+SIGN END
      ELSE IF TYP=T THEN BEGIN ADNUM; NET:=NET+SIGN END
```

```
        ELSE BEGIN ERROR(33); GOTO 1 END
      UNTIL ENDCHAR(CH);

      *IF EXTERNAL SYMBOL, IT MUST APPEAR ALONE*
      IF (TYP=EXE)^(COUNT#1) THEN BEGIN ERROR(34); GOTO 1 END;
      IF (NET<0)v(NET>1) THEN  BEGIN ERROR(48); GOTO 1 END;
      IF NET=0 THEN TYP:=E E;
   END;

   *CHECK RANGE*
   IF ABSPROG THEN
         BEGIN
            IF  (OPN<14)^(NUM>20000B) THEN
            IF (LC DIV 20000B)#(NUM DIV 20000B) THEN ERROR(47)
         END
   ELSE *RELOCATABLE PROG.*
         BEGIN
            IF (TYP=E E)^(NUM>20000B) THEN
            IF (OPN<14)v(OPN=87)v((OPN>78)^(OPN<81)) THEN ERROR(35);
         END;
   IF NUM<0 THEN BEGIN NUM:=NUM+2000000B; IF OPN#82 THEN ERROR(36) END
   ELSE BEGIN
         IF OPN>15 THEN
         IF OPN<27 THEN *I/O*  BEGIN IF NUM>778 THEN ERROR(41) END
         ELSE IF OPN<33 *EAU* THEN BEGIN IF NUM>178 THEN ERROR(42) END
         ELSE *EXT. ARITH AND PSEUDO* IF NUM>777778 THEN ERROR(43)
      END;

   IF CH=E,E THEN
   BEGIN
      IF SCANTOCOMMA THEN BEGIN SCANTOCOMMA:=FALSE; GOTO 1 END
                     ELSE NEXTCHAR(CH);
      IF CH=EIE THEN BEGIN INDIRECT:=TRUE;
                        IF (OPN#87) THEN
                        IF (OPN>14)^(OPN<74)v(OPN>87) THEN ERROR(29)
                     END ELSE
      IF CH=ECE THEN BEGIN
                        IF (OPN<15)v(OPN>36) THEN ERROR(29) ELSE
                        IF NOT(OPN IN CSET) THEN ERROR(29);
                        APPEND(NUM,0,13)
                     END
      ELSE BEGIN ERROR(5); ERROR(29) END;
   END;
!! 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 INTEXT(1)#E E 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 S) AS NEGATIVE NUMBERS IN PASS ONE TO DISTINGUISH THEM *
 FROM ORDINARY INSTRUCTIONS*

LABEL 1;
VAR
   I,SI,K,P,G,KL,PR,GP,NUM,CODE : INTEGER;
   FIRST : BOOLEAN;
   SAVE : ARRAY [1..2] OF INTEGER;

{-------------------------------------------------------------------}
PROCEDURE ERREXIT(N:INTEGER);
BEGIN
   ERROR(N); WORD:=0; WHILE S*.OPFLD<0 DO GET(S); GOTO EXIT 1
END *ERREXIT*;
{-------------------------------------------------------------------}

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

FUNCTION KLASS(NUM:INTEGER):INTEGER;

*CHECK TO SEE IF THE OP BELONGS TO GROUP A OR B OR EITHER OF THE TWO:  *
   GP A - KLASS=1, GP B - KLASS=2, EITHER - KLASS=0*

VAR I:INTEGER;
BEGIN
   I:=0; REPEAT I:=I+1 UNTIL NUM<=JT[I];
   KLASS:=KT[I]
END *KLASS*;
{-------------------------------------------------------------------}

FUNCTION GROUP(NUM:INTEGER):INTEGER;

* CHECK IF OP BELONGS TO SHIFT-ROTATE GROUP OF ALTER-SKIP GROUP OR BOTH.*
   SRG - GROUP=1, ASG - GROUP=2, BOTH - GROUP=0*

BEGIN
   IF NUM<54 THEN GROUP:=1
   ELSE IF NUM<57 THEN GROUP:=0
   ELSE GROUP:=2
END *GROUP*;
{-------------------------------------------------------------------}

FUNCTION PREC(NUM:INTEGER):INTEGER;
```

```
* CHECK PRECEDENCE OF OP *

BEGIN
   IF NUM<54 THEN BEGIN IF FIRST THEN PREC:=0 ELSE PREC:=6 END
             ELSE PREC:=PT[NUM]
END *PREC*;
*-----------------------------------------------------------------------*

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 *
 AN OP*

VAR N : INTEGER;
BEGIN
   CASE GP OF
      0 : BEGIN N:=0; SI:=SI+1; SAVE[SI]:=NUM END;
      1 : IF (~FIRST)^(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*
   WORD:=0; NUM:=-OPN; ERRCHECK;
   FIRST:=TRUE; SI:=0; SAVE[1]:=0; SAVE[2]:=0;

   *DETERMINE KLASS, GROUP AND PREC OF THE FIRST OP IN A MULTIPLE INSTR*
   KL:=KLASS(NUM); GP:=GROUP(NUM); PR:=PREC(NUM);

   *PUT THE CODE OF THE FIRST OP IN WORD*
   CODE:=GETCODE(NUM,GP); APPEND(WORD,0,CODE); FIRST:=FALSE;

   GET(S); OPN:=S*.OPFLD;
   WHILE OPN<0 DO
   BEGIN *PROSESS REMAINING OPS, CHECK COMPATIBILITY IN ALL THREE WAYS* *
      NUM:=-OPN; ERRCHECK;

      K:=KLASS(NUM);
      IF KL=0 THEN KL:=K ELSE IF (K≠KL)^(K≠0) THEN ERREXIT(14);

      G:=GROUP(NUM);
      IF GP=0 THEN GP:=G ELSE IF (G≠GP)^(G≠0) THEN ERREXIT(15);

      P:=PREC(NUM);
      IF PR<P THEN PR:=P ELSE ERREXIT(16);

      CODE:=GETCODE(NUM,GP); APPEND(WORD,0,CODE);
      GET(S); OPN:=S*.OPFLD;
   END;

   IF SAVE[1]≠0 THEN
      BEGIN
         IF GP=0 THEN GP:=1;
         FOR I:=1 TO 2 DO IF SAVE[I]≠0 THEN
                  BEGIN CODE:=GETCODE(SAVE[I],GP); APPEND(WORD,0,CODE) END*
     END;
```

```
!!
END *MULTINST*;
*-----------------------------------------------------------------------*

PROCEDURE PASSONE;

LABEL 2,3;
VAR
   FLAG,FIRSTHED,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 ABSPROG THEN BEGIN ERROR(6); GOTO EXIT 2 END
END *CHECKRELOC*;
*-----------------------------------------------------------------------*

PROCEDURE GETXIF;

*IF AN N (OR Z) OPTION IS CHOSEN THEN THIS PROCEDURE IS CALLED TO SKIP
 ALL STATEMENTS BETWEEN AN IFZ (OR IFN) AND XIF PSEUDO*


   VAR A:ALFA; F:BOOLEAN;
BEGIN
   F:=FALSE; NZFLAG:=FALSE;
   REPEAT
      READLINE; COL:=1; SKIPTOBLANK; GETNAME(A);
      IF (A=='IFN')v(A=='IFZ') THEN ERROR(45);
      IF A=='XIF' THEN BEGIN S*.OPFLD:=108; F:=TRUE END
      ELSE IF A=='END' THEN BEGIN S*.OPFLD:=89; F:=TRUE END
      ELSE BEGIN S*.OPFLD:=0; PUT(S) END
   UNTIL F
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;
1: REALCONST:=FALSE;
   REPEAT
      NEXTCHAR(CH);
      IF (CH=='E')v(CH=='.') THEN REALCONST:=TRUE
   UNTIL ENDCHAR(CH);

   CASE OPN OF
  *OEC* 86 : IF REALCONST THEN INCRCOUNT(2) ELSE INCRCOUNT(1);
```

```
  *DEX* 88 : INCRCOUNT(3);
  *OCT* 98 : INCRCOUNT(1);
   END;

   IF CH=',' THEN GOTO 1
END *COUNTCONST*;
{-------------------------------------------------------------------}

PROCEDURE INCREMENTLOC;

*ENTER LABEL AND INCREMENT LOC COUNTER BY 1 IF AN ORDINARY INSTRUCTION, *
 INCREMENT LOC COUNTER BY REPN IF THE INSTRUCTION WAS FOLLOWED BY A REP *
 PSEUDO*
BEGIN
   IF REPFLAG THEN BEGIN
                     REPFLAG:=FALSE; INCRCOUNT(REPN);
                     IF LBL<>EMPTY THEN ERROR(28)
                   END
   ELSE BEGIN ENTERLABEL; INCRCOUNT(1) END
END *INCREMENTLOC*;
{-------------------------------------------------------------------}

BEGIN
   SCAN1:=TRUE; REPFLAG:=FALSE; FIRSTHED:=TRUE; SUPFLAG:=FALSE;
   FLAG:=TRUE; PAGEFLAG:=TRUE; FIRSTERR:=TRUE; SCANTOCOMMA:=FALSE;
   BPC:=0; SAVELC:=0; XT:=0; NT:=0; COM:=0; ER:=0; NZFLAG:=FALSE;
   LINES(3);

1: *PROCESS THE NEXT SOURCE LINE*
   READLINE; GETLABEL;
   IF CH=EOL THEN *BLANK CARD* BEGIN OPN:=0; STORELINE; GOTO 2 END
             ELSE GETOP(OPN);

*CHECK TO SEE THAT THE FIRST VALID OPCODE IN AN ABSOLUTE PROGRAM IS
 FOLLOWED BY AN ORG STATEMENT AND IN A RELOCATABLE PROGRAM BY A NAM
 STATEMENT*
   IF FLAG THEN IF OPN<85 THEN ERROR(39)
                ELSE IF NOT((OPN-85) IN PSEUDOSUBSET) THEN ERROR(39);

   IF OPN=0 THEN *ILLEGAL OPERATION* BEGIN ERROR(3); STORELINE END
   ELSE IF OPN<0 THEN *MULTIPLE INSTRUCTION* INCREMENTLOC
   ELSE
   BEGIN STORELINE;
      IF OPN<73 THEN * MACHINE INSTRUCTION *  INCREMENTLOC
      ELSE IF OPN<81 THEN *EXTENDED ARITH. INSTRUCTION*
           BEGIN
              IF REPFLAG THEN
              BEGIN
                 REPFLAG:=FALSE; INCRCOUNT(2*REPN);
                 IF LBL<>EMPTY THEN ERROR(28)
              END ELSE BEGIN ENTERLABEL; INCRCOUNT(2) END;

              IF OPN>78 THEN  *FAD,FSB,FMP,FDV*
              BEGIN
                 CHECKRELOC;
                 A:=ELBL[OPN]; SEARCH(A,N,CH);
                 IF NOT PRESENT THEN BEGIN
                                       XT:=XT+1; ENTER(A,XT,EXE); EXT[XT]:=A*
                                     END

              END
           END
```

```
  ELSE { PSEUDO-INSTRUCTION }
  CASE OPN OF
{ARS} 82 : BEGIN ENTERLABEL; INCRCOUNT(1) END;
{ASC} 83 : BEGIN
             ENTERLABEL; SCANTOCOMMA:=TRUE; SCANOPERAND(N,TYP);
             IF (N>1)&(N<29) THEN INCRCOUNT(N) ELSE ERROR(9)
           END;
{BSS} 84 : BEGIN ENTERLABEL; SCANOPERAND(N,TYP); INCRCOUNT(N) END;
{COM} 85 : BEGIN
             CHECKRELOC;
             REPEAT
               N:=0; GETNAME(A);
               IF CH='(' THEN BEGIN
                                NEXTCHAR(CH); SKIPBLANKS;
                                GETNUM(N); SKIPBLANKS;
                                IF CH=')' THEN NEXTCHAR(CH)
                                          ELSE SKIPREST
                              END
               ELSE IF ENDCHAR(CH) THEN N:=1
               ELSE SKIPREST;
               ENTER(A,COM,'C'); COM:=COM+N; COL:=COL+1
             UNTIL BLANKOREOL(CH)
           END;

{DEC} 86 : COUNTCONST(OPN);
{DEF} 87 : BEGIN ENTERLABEL; INCRCOUNT(1) END;
{DEX} 88 : COUNTCONST(OPN);
{END} 89 : BEGIN
             ENDLC:=LC; S^.LN:=-1; NLN:=LN; PUT(S);
             SKIPBLANKS; TRNSFR[1]:=0; {R-BIT}
             IF CH=EOL THEN  {NO TRANSFER ADDRESS SPECIFIED}
                       BEGIN TRNSFR[2]:=0 {T-BIT}; TRNSFR[3]:=0 END
             ELSE BEGIN { EVALUATE TRANSFER ADDRESS}
                    TRNSFR[2]:=1; SCANOPERAND(TRNSFR[3],TYP);
                    IF TYP='B' THEN TRNSFR[1]:=2;
                  END;
             GOTO 3
           END;
{ENT} 90 : BEGIN
             CHECKRELOC;
             REPEAT
               NT:=NT+1;
               IF NT>MNT THEN BEGIN  ERROR(27); GOTO 2 END;
               GETNAME(ENT[NT].SYM);
               IF ~ENDCHAR(CH) THEN SKIPREST; COL:=COL+1;
             UNTIL BLANKOREOL(CH)
           END;
{EQU} 91 : BEGIN
             SCANOPERAND(NUM,TYP);
             ENTER(LBL,NUM,TYP)
           END;
{EXT} 92 : BEGIN
             CHECKRELOC;
             REPEAT
               GETNAME(A); XT:=XT+1;
               IF XT>MXT THEN BEGIN  ERROR(26); GOTO 2 END;
               ENTER(A,XT,'X'); EXT[XT]:=A;
               IF ~ENDCHAR(CH) THEN SKIPREST; COL:=COL+1
             UNTIL BLANKOREOL(CH)
           END;
{MED} 93 : S^.LN:=-1;
```

```
    ^IFN^ 94 : BEGIN
                  PUT(S); IF NZFLAG THEN ERROR(45) ELSE NZFLAG:=TRUE;    *
                  IF OPTION="NE" THEN GOTO 1 ELSE GETXIF
               END;
    ^IFZ^ 95 : BEGIN
                  PUT(S); IF NZFLAG THEN ERROR(45) ELSE NZFLAG:=TRUE;    *
                  IF OPTION="ZE" THEN GOTO 1 ELSE GETXIF
               END;
    ^LST^ 96 : ;
    ^NAM^ 97 : BEGIN
                  IF ABSPROG THEN ERROR(39);
                  FLAG:=FALSE; CHECKRELOC; SKIPBLANKS;
                  IF CH=EOL THEN PROG:=EMPTY ELSE GETNAME(PROG);
                  SKIPBLANKS;
                    IF CH="," THEN BEGIN NEXTCHAR(CH); GETNUM(NMN) END   *
                             ELSE NMN:=0;
               END;
    ^OCT^ 98 : COUNTCONST(OPN);
    ^ORB^ 99 : BEGIN CHECKRELOC; PAGEFLAG:=FALSE; ^BASE PAGE^ END;
    ^ORG^ 100: BEGIN
                  IF ~ABSPROG ^ FLAG THEN ERROR(39) ELSE FLAG:=FALSE;    *
                  PAGEFLAG:=TRUE; ^CURRENT PAGE^
                  SCANOPERAND(NUM,TYP);
                  IF ~ABSPROG^(TYP#"E") THEN BEGIN ERROR(30); GOTO 2 END*
                  ELSE LC:=NUM;
                  IF SAVELC=0 THEN SAVELC:=LC
               END;
    ^ORR^ 101: BEGIN
                  PAGEFLAG:=TRUE; ^CURRENT PAGE^
                  IF SAVELC#0 THEN BEGIN LC:=SAVELC; SAVELC:=0 END
               END;
    ^REP^ 102: BEGIN
                   ENTERLABEL;
                   REPFLAG:=TRUE; SCANOPERAND(REPN,TYP);
                   IF TYP#" " THEN ERROR(39);
               END;
    ^SKP^ 103: S^.LN:=-1;
    ^SPC^ 104: S^.LN:=-1;
    ^SUP^ 105: ;
    ^UNL^ 106: ;
    ^UNS^ 107: ;
    ^XIF^ 108: NZFLAG:=FALSE;
    ^SWP^ 109: BEGIN ENTERLABEL; INCRCOUNT(1) END;
      END
  END;
2: PUT(S); GOTO 1;

3: PRINTERR; LINES(3);

  ^CHECK FOR ENTRY LABELS AND RELATED ERRORS^
  FOR I:=1 TO NT DO
  BEGIN
     SEARCH(ENT[I].SYM,N,TYP);
     IF PRESENT THEN BEGIN
                        IF (TYP="C")v(TYP="X") THEN ERROR(24)
                        ELSE BEGIN ENT[I].LOC:=N; ENT[I].TYP:=TYP END    *
                     END
                ELSE ERROR(25);
     IF NER#0 THEN BEGIN
                     WRITE(ENT[I].SYM); LINES(1); PRINTERR
                   END
```

```
   END;
   LINES(4);
   WRITE(=** =); IF ER=0 THEN WRITE(= NO=) ELSE WRITE(ER:5);
   WRITE(= ERRORS DETECTED IN PASS 1 **=); 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,MP)
   END
END *REPINSTR*;

*=============================================================*

PROCEDURE PASSTWO;
BEGIN
   TITLE;
   ER:=0; SPC:=0; SCAN1:=FALSE; PAGEFLAG:=TRUE; SUPFLAG:=FALSE;
   RESET(S);

1: LN:=S*.LN; LC:=S*.LOCFLD; OPN:=S*.OPFLD;
   IF LSTFLAG THEN IF LN=-1 THEN WRITE(EOL,=*=) ELSE WRITE(LN,= =,= =); *

   IF OPN=0 THEN *COMMENT CARD*
      BEGIN SPACE( 20); WRITELINE; GET(S); GOTO 1 END;
   EXPAND(S*.LINE,INTEXT);
   EXTRALINE:=FALSE; MP:=-1;
   IF OPN<0 THEN *MULTIPLE INSTRUCTION*
      BEGIN
         MULTINST(WORD);
         STOREWORD(LC,WORD,= =,MP);
         IF REPFLAG THEN REPINSTR;
      END *MULTIPLE INSTRUCTION*

   ELSE *OPN>0*
   IF OPN<73 THEN * MACHINE INSTRUCTION *
   BEGIN
      CODE:=OPCODE[OPN];
      IF OPN<33 THEN *OPERAND EXPECTED*
         BEGIN
            SCANOPR2(WORD,TYP); IF INDIRECT THEN CODE:=CODE+100000B;
            IF ABSPROG THEN
                     BEGIN
                        IF WORD>=2000B THEN
                        BEGIN
                           APPEND(CODE,0,2000B); WORD:=WORD MOD 2000B
                        END
                     END
            ELSE BEGIN *RELOC. PROG.*
                     IF (TYP==R=)*(TYP==X=)*(TYP==C=) THEN
                                          APPEND(CODE,0,2000B);
                     IF (OPN<=14)*((TYP==R=)*(TYP=="B=)*(TYP=="C=)) THEN
```

```
                BEGIN *MEM. REF.*.MR:=WORD; WORD:=CODE; GOTO 99 END*
            END;
          APPEND(WORD,0,CODE)
        END
        ELSE *NO OPERAND* BEGIN WORD:=CODE; TYP:=Ξ Ξ END;
  99: STOREWORD(LC,WORD,TYP,MR);
  IF REPFLAG THEN REPINSTR;
END *MACHINE INSTRUCTION*
ELSE IF OPN<81 THEN *EXTENDED ARITH. INSTRUCTION*
  BEGIN
    IF OPN<77 THEN *MPY,DIV,DLD,DST*
      BEGIN
        WORD:=OPCODE[OPN]; TYP:=Ξ Ξ
      END
    ELSE *FAD,FSB,FMP,FDV*
      BEGIN
        IF ABSPROG THEN ERROR(6);
        SEARCH(FLBL[OPN],N,TYP);
        WORD:=160000B; *JSB, CURRENT PAGE*
        APPEND(WORD,0,N)
      END;

    STOREWORD(LC,WORD,TYP,MR); EXTRALINE:=TRUE;
    SCANOPR2(N,T); IF INDIRECT THEN N:=N+100000B;
    STOREWORD(LC+1,N,T,MR);
    IF REPFLAG THEN
    BEGIN
      REPFLAG:=FALSE;
      FOR I:=1 TO REPN-1 DO
      BEGIN
        LC:=LC+2; SPACE(12);
        EXTRALINE:=FALSE; STOREWORD(LC,WORD,TYP,MR);
        EXTRALINE:=TRUE ; STOREWORD(LC+1,N,T,MR)
      END
    END
  END *EXT. ARITH. INSTR.*
ELSE *PSEUDO-OPERATION*
BEGIN
  CASE OPN OF

 *ABS* 82 : BEGIN
              SCANOPR2(N,TYP); IF TYP≠Ξ Ξ THEN ERROR(31);
              STOREWORD(LC,N,TYP,MR);
            END;
 *ASC* 83 : BEGIN
              SCANTOCOMMA:=TRUE; SCANOPR2(NUM,TYP);
              IF (NUM<1)∨(NUM>20) THEN BEGIN ERROR(9); COPYCARD END*
              ELSE BEGIN
                     GETASC(N); STOREWORD(LC,N,Ξ Ξ,MR);
                     LC:=LC+1;  EXTRALINE:=TRUE;
                     FOR I:=1 TO NUM-1 DO
                     BEGIN
                       GETASC(N); STOREWORD(LC,N,Ξ Ξ,MR);
                       LC:=LC+1
                     END
                   END
            END;
 *BSS* 84 : IF LSTFLAG THEN BEGIN
                              OUTOCT(LC); OUTOCT(0);
                              SPACE(6); WRITELINE
                            END;
```

```
^COM+ 85 : COPYCARD;
^DEC+ 86 : BEGIN
             MOVECOL;
             REPEAT
               GETCONST(KIND,X,N);
               CASE KIND OF
                    0 : BEGIN ERROR(20); STOREINT(N) END;
                    1 : STOREINT(N);
                    2 : STOREREAL(SINGLE,X,N)
               END;
               CURRCHAR(CH); COL:=COL+1
             UNTIL BLANKOREOL(CH)
           END;
^DEF+ 87 : BEGIN SCANOPR2(N,TYP); IF INDIRECT THEN N:=N+100000B;
             STOREWORD(LC,N,TYP,MR)
           END;
^DEX+ 88 : BEGIN
             MOVECOL;
             REPEAT
               GETCONST(KIND,X,N);
               CASE KIND OF
                    0 : ERROR(20);
                    1 : BEGIN X:=N; N:=0 END;
                    2 : ;
               END;
               STOREREAL(DOUBLE,X,N);
               CURRCHAR(CH); COL:=COL+1
             UNTIL BLANKOREOL(CH)
           END;
^END+ 89 : GOTO 2;
^ENT+ 90 : COPYCARD;
^EQU+ 91 : IF LSTFLAG THEN
           BEGIN
             SCANOPR2(N,TYP); OUTOCT(N); SPACE(13); WRITELINE
           END;
^EXT+ 92 : COPYCARD;
^HED+ 93 : BEGIN
             I:=0; MOVECOL; COL:=COL-1; J:=73-COL;
             REPEAT
                       I:=I+1; HDNG[I]:=INTEXT[COL+I]
             UNTIL (HDNG[I]=EOL)∨(I≥J);
             HDNG[I]:=EOL;
             TITLE
           END;
^IFN+ 94 : COPYCARD;
^IFZ+ 95 : COPYCARD;
^LST+ 96 : IF LSTOUTPUT THEN
             BEGIN
               LSTFLAG:=TRUE; WRITE(EOL,Ξ Ξ,LN,Ξ Ξ,Ξ Ξ); COPYCARD
             END;
^NAM+ 97 : IF LSTFLAG THEN
             BEGIN OUTOCT(0); SPACE(13); WRITELINE END;
^OCT+ 98 : BEGIN
             MOVECOL;
             REPEAT
               GETOCT(N); STOREINT(N);
               IF ~ENDCHAR(CH) THEN SKIPREST; COL:=COL+1
             UNTIL BLANKOREOL(CH)
           END;
^ORB+ 99 : BEGIN
             PAGEFLAG:=FALSE;
```

```
                OPYCARD
              END;
    *ORG*  100: BEGIN
                 PAGEFLAG:=TRUE;
                 IF LSTFLAG THEN
                 BEGIN
                   SCANOPR2(N,TYP); OUTOCT(N); SPACE(13); WRITELINE  *
                 END
              END;
     *OPR*  101: BEGIN PAGEFLAG:=TRUE;  COPYCARD END;
     *REP*  102: BEGIN SCANOPR2(REPN,TYP); REPFLAG:=TRUE; COPYCARD END;  *
     *SKP*  103: TITLE;
     *SPC*  104: BEGIN  SCANOPR2(N,TYP); LINES(N)  END;
     *SUP*  105: BEGIN SUPFLAG:=TRUE; COPYCARD END;
     *UNL*  106: BEGIN COPYCARD; LSTFLAG:=FALSE END;
     *UNS*  107: BEGIN SUPFLAG:=FALSE; COPYCARD END;
     *XIF*  108: COPYSEQ;
     *SWP*  109: STOREWORD(LC,101100B,"  ",MR);
       END
  END;
  GET(S); GOTO 1;

2: *STORE LITERALS*
  EXTRALINE:=TRUE;
  FOR I:=1 TO HLT DO IF LITTAB[I,1]=NOLIT THEN GOTO 3
                    ELSE STOREWORD(LITTAB[I,2],LITTAB[I,1],"  ",-1);  *

3: IF LSTFLAG THEN BEGIN WRITE(NLN); SPACE(22); WRITELINE END; LINES(4);*

  WRITE("** ");  IF ER=0 THEN WRITE("NO") ELSE WRITE(ER:5);
  WRITE(" ERRORS DETECTED IN PASS 2 **"); LINES(3);

END *PASSTWO*;

*=============================================================================*

PROCEDURE PUNCHCARD(A:ARRINT16);

*TRANSFER 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,M,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   *
```

```
 ARE MADE TRANSFER BF ON PUCH FILE AND START AGAIN.  (INVERSION MEANS   *
 TURNING A CARD COLUMN UPSIDE DOWN)*

BEGIN
      Z:=Z+1;
      APPEND(BF,12,INVERT[N]);
      IF Z=5 THEN BEGIN P*:=BF; PUT(P); BF:=0; Z:=0; C:=C+5 END
END *PUSH*;
-------------------------------------------------------------------------

PROCEDURE STORE(N:INTEGER);

VAR K:INTEGER;
BEGIN
      IF PUNCHBIN THEN *BREAK HP WORD AT 8TH BIT AND SEND FOR PUNCHING*
      BEGIN
         K:=N DIV 400B; PUSH(K);
         K:=N-K*400B; PUSH(K)
      END;
      IF LISTBIN THEN *PRINT HP WORD IN OCTAL FORMAT*
      BEGIN
         M:=M+1; OUTOCT(N); WRITE(=  =);
         IF M=15 THEN BEGIN M:=0; LINES(1) END;
      END
END *STORE*;
-------------------------------------------------------------------------

BEGIN
   BF:=0; Z:=0; M:=0;
   RESET(R);

   WHILE NOT EOF(2) DO
   BEGIN *NEW RECORD EACH LOOP*
      W[2]:=R*.LOC;        *ABS LOAD ADDRESS*
      CHECKSUM:=R*.LOC;    *INITIALIZE CHECKSUM*
      LAST:=R*.LOC-1;

      C:=0;
      L:=2;
      REPEAT
         IF R*.LOC=LAST+1 THEN LAST:=LAST+1 ELSE GOTO 1;
         L:=L+1;
         W[L]:=R*.WORD;    *STORE INSTRUCTION*
         CHECKSUM:=CHECKSUM+R*.WORD;    *UPDATE CHECKSUM*
         GET(R)
      UNTIL (L=39)+EOF(R);

   1: *COMPLETE RECORD AND TRANSFER TO FILE P*
      I:=L-2;          *RECORD LENGTH*
      W[1]:=0; INSERT(I,5,W[1]);
      W[L+1]:=CHECKSUM MOD 2**8;
      IF LISTBIN 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 ZEROES*
   WHILE C<80 DO BEGIN P*:=0; PUT(P); C:=C+5 END
   END;
END *PUNCHABSBIN*;
```

```
{=============================================================}

PROCEDURE PUNCHRECORD(RECNAME:ALFA; RL:INTEGER);

{PUNCH (AND/OR PRINT) ONE RELOCATABLE BINARY RECORD (60 HP WORDS)}

VAR I,N,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; OUTOCT(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:=C; K:=1;
         FOR I:=1 TO 4 DO
         BEGIN
            N:=REM;
            FOR J:=C+1 TO C+3 DO APPEND(N,16,W[J]);
            K:=K+4;
            IF K<16 THEN BEGIN
                           C:=C+4;
                           J:=W[C] DIV D[I];
                           REM:=W[C]-J*D[I];
                           APPEND(N,16-K,J)
                        END
                    ELSE C:=C+3;
            P+1:=N; PUT(P)
         END
      UNTIL C=60;
   END
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}
BEGIN
   CS:=W[2];
   FOR I:=4 TO RL DO CS:=CS+W[I];
   CHECKSUM:= CS MOD 2000008
END {CHECKSUM};

{=============================================================}

PROCEDURE REPACK(A:ALFA; VAR B:ARRINT3);
```

```
*THE NAME A (5 CHARS) IS RE-STORED 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; C:ARRCHAR10;
BEGIN
   UNPACK(A,C,1); C[6]:=EOL;
   FOR I:=1 TO 3 DO
   BEGIN
      J:=2*I-1;
      B[I]:=ASCIICODE[C[J]];
      N:=ASCIICODE[C[J+1]];
      APPEND(B[I],8,N)
   END
END *REPACK*;

*=============================================================================*

PROCEDURE NAMRECORD;

*FORM A NAM RECORD IN THE ARRAY W[1..6]*

VAR P:ARRINT3;  *GLOBALS-- W,PROG,ENDLC,BPC,COM*
BEGIN
   W[1]:=104008;           *RECORD LENGTH=17, LEFT-SHIFTED 8 BITS*
   W[2]:=200008;           *IDENT*
   REPACK(PROG,P);         *W[4..6] TO CONTAIN PROG NAME*
   APPEND(P[3],0,4008);
   FOR I:=1 TO 3 DO W[I+3]:=P[I];
   W[7]:=ENDLC;            *LENGTH OF MAIN PROG SEGMENT*
   W[8]:=BPC;              *LENGTH OF BASE PAGE SEGMENT*
   W[9]:=COM;              *LENGTH OF COMMON SEGMENT*
   W[10]:=NMN;             *NUMERAL PARAMETER IN NAM STATEMENT*
   IF NMN=0 THEN W[11]:=0 ELSE W[11]:=1438;
   FOR I:=12 TO 17 DO W[I]:=0;
   W[3]:=CHECKSUM(17);     *CHECKSUM*
   PUNCHRECORD('NAM RECORD',17)
END *NAMRECORD*;

*=============================================================================*

PROCEDURE ENTRECORD;

*FORM ENTRY RECORD*

VAR I,J,K,RL:INTEGER; P:ARRINT3;  *GLOBALS-- W,NT,ENT*

BEGIN
   RL:=4*NT+3;             *RECORD LENGTH*
   W[1]:=0; INSERT(RL,8,W[1]);
   W[2]:=400008+NT;        *IDENT+ENTRIES*
   K:=3;
   FOR I:=1 TO NT DO
   BEGIN
      REPACK(ENT[I].SYM,P);
      IF ENT[I].TYP='RE' 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;
```

```
   W[3]:=CHECKSUM(RL);    *CHECKSUM*
   PUNCHRECORD('ENT RECORD',RL)
END *ENTRECORD*;

{*************************************************************}

PROCEDURE EXTRECORD;

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

VAR I,J,K,RL,M1,M2:INTEGER; MORE:BOOLEAN; P:ARRINT3;
   *GLOBALS-- W,XT,EXT*
BEGIN
   M1:=1;
   IF XT<=19 THEN BEGIN MORE:=FALSE; M2:=XT END
             ELSE BEGIN MORE:=TRUE;  M2:=19 END;
1: *NEW EXT RECORD*
   RL:=3*M2+3;              *RECORD LENGTH*
   W[1]:=0; INSERT(RL,8,W[1]);
   W[2]:=M2+10000C09;       *ENTRIES+IDENT*
   K:=3;
   FOR I:=M1 TO M2 DO
   BEGIN
      REPACK(EXT[I],P); APPEND(P[3],C,I);
      FOR J:=1 TO 3 DO W[K+J]:=P[J];
      K:=K+3
   END;
   W[3]:=CHECKSUM(RL);    *CHECKSUM*
   PUNCHRECORD('EXT RECORD',RL);

   IF MORE THEN BEGIN
                  M1:=M2+1; M2:=M2+20;
                  IF M2>XT THEN BEGIN M2:=XT; MORE:=FALSE END;
                  GOTO 1
                END;
END *EXTRECORD*;

{*************************************************************}

PROCEDURE DBLRECORDS(VAR F:FILEOUTCODE; PG:BIT);
*THE DBL 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,N,NI,LAST : INTEGER;
   G,ENDF : BOOLEAN;
   *GLOBAL -- W*

BEGIN
   RESET(F); ENDF:=FALSE;
   REPEAT *NEW RECORD EVERY LOOP*
      L:=4; NI:=0; G:=TRUE;
      LAST:=F*.LOC-1;
      W[4]:=F*.LOC;  *RELOCATABLE LOAD ADDRESS*
      WHILE (L<58)^(^ENDF)^G DO
      BEGIN
         I:=0; L:=L+1; J:=L; W[J]:=0;
         *W[J] TO CONTAIN RELOCATION INDICES FOR UPTO NEXT FIVE WORDS*
         REPEAT
            IF F*.LOC=LAST+1 THEN LAST:=LAST+1
```

```
                                ELSE BEGIN G:=FALSE; GOTO 3 END;
            I:=I+1; L:=L+1;
            IF F^.MREF>0 THEN (*MEMORY REFERENCE INSTR, USES TWO WORDS*)
               BEGIN
                  APPEND(W[J],3,5);
                  W[L]:=F^.WORD;
                  CASE F^.TYP OF
                     'R': K:=0;   'B': K:=1;   'C': K:=2
                  END;
                  APPEND(W[L],C,K);
                  L:=L+1; W[L]:=F^.MREF
               END
            ELSE (*NO MEMORY REFERENCE*)
               BEGIN
                  W[L]:=F^.WORD;
                  CASE F^.TYP OF
                     ' ': K:=0;
                     'R': K:=1;
                     'B': K:=2;
                     'C': K:=3;
                     'X': K:=4
                  END;
                  APPEND(W[J],3,K)
               END;
            GET(F); IF EOF(F) THEN ENDF:=TRUE;
         UNTIL (I=5)+ENDF+(L>59);
      3: APPEND(W[J],(5-I)*3+1,0);
         NI:=NI+I;
      END;

      W[1]:=0; INSERT(L,8,W[1]);  (*RECORD LENGTH*)
      W[2]:=6*2**8+PG*100+8+NI;   (*IDENT,ENTRIES, ETC.*)
      W[3]:=CHECKSUM(L);          (*CHECKSUM*)
   PUNCHRECORD('OBL RECORD',L)
   UNTIL ENDF;
END (*OBLRECORDS*);

(*=============================================================*)

PROCEDURE ENDRECORD;

VAR I:INTEGER;  (*GLOBALS-- W,TRNSFR*)
BEGIN
   W[1]:=2000B;                (*RECORD LENGTH=4, SHIFTED 8 BITS TO LEFT*)
   W[2]:=12000B+TRNSFR[1]+TRNSFR[2];
   W[4]:=TRNSFR[3];
   W[3]:=(W[2]+W[4]) MOD 200000B;
   PUNCHRECORD('END RECORD',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,D1,D2 : INTEGER;  A:ALFA; C:ARRCHAR10;

(*-------------------------------------------------------------*)
```

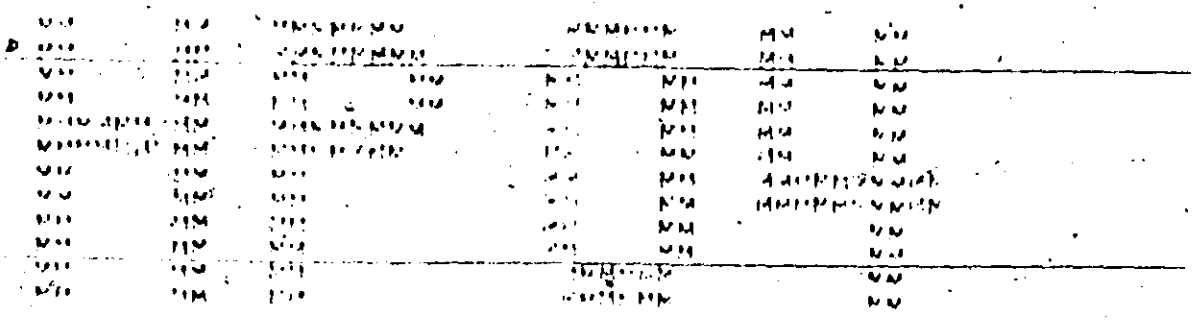
```
PROCEDURE BLANKLINES(N:INTEGER);
VAR I:INTEGER;
BEGIN
   WRITE(EOL,Ξ Ξ);
   FOR I:=1 TO N DO
   BEGIN
      SPACE(37); OUTCH(Ξ*Ξ); SPACE(60); WRITE(Ξ*Ξ,EOL,Ξ Ξ)
      END;
   SPACE(37)
END ↑BLANKLINES↓;
↑----------------------------------------------------------------↓*

PROCEDURE CENTRE(A:ARRCHAR90; N1,N2:INTEGER);
VAR I,J,K,PAD : INTEGER;
BEGIN
   J:=N1; K:=N2;
   WHILE (A[J]=Ξ Ξ)∧(J<N2) DO J:=J+1;
   IF J≠N2 THEN WHILE A[K]=Ξ Ξ DO K:=K-1;
   PAD:=(80-K+J) DIV 2;
   OUTCH(Ξ*Ξ); SPACE(PAD); IF ~ODD(K-J) THEN PAD:=PAD-1;
   FOR I:=J TO K DO OUTCH(A[I]); SPACE(PAD); OUTCH(Ξ*Ξ);
END ↑CENTRE↓;

↑----------------------------------------------------------------↓*

BEGIN
   D1:=NPROG DIV 10; D2:=NPROG MOD 10;

   ↑PUNCH HEADER CARD↓
   HEDCARD[10]:=IMAGE[2*D1];  HEDCARD[11]:=IMAGE[2*D1+1];
   HEDCARD[12]:=IMAGE[2*D2];  HEDCARD[13]:=IMAGE[2*D2+1];
   PUNCHCARD(HEDCARD);

   ↑PRINT TITLE PAGE↓
   LINES(12);
   FOR I:=1 TO 12 DO
   BEGIN
      SPACE(45); WRITE(HH[I]      ,Ξ  Ξ); WRITE(PP[I],Ξ   Ξ);
      WRITE(DIG[D1,I]); WRITE(Ξ  Ξ,DIG[D2,I],EOL,Ξ Ξ)
   END;
   J:=0; REPEAT J:=J+1; HDNG[J]:=INTEXT[J+7] UNTIL (HDNG[J]=EOL)∨(J=61);*
   HDNG[J]:=EOL;
   I:=0; REPEAT I:=I+1 UNTIL INTEXT[I]=EOL;
   REPEAT INTEXT[I]:=Ξ Ξ; I:=I+1 UNTIL I>80;
   FOR I:=1 TO 7 DO C[I]:=INTEXT[I];
   FOR I:=8 TO 10 DO C[I]:=Ξ Ξ; PACK(C,1,A);
   IF A≠HPASMA,Ξ    THEN ERROR(46);
   LINES(5); SPACE(37); MULTIPRINT(Ξ*Ξ,62); BLANKLINES(3);
WRITE(Ξ*            HEWLETT-PACKARD ASSEMBLY LANGUAGE PROGRAM          *Ξ);*
   BLANKLINES(3);
WRITE(Ξ*                                TITLE:                         *Ξ);*
   BLANKLINES(2); CENTRE(INTEXT, 8,68); BLANKLINES(3);
WRITE(Ξ*                              PROGRAMMER:                      *Ξ);*
   BLANKLINES(2); CENTRE(INTEXT,69,80); BLANKLINES(3);
   MULTIPRINT(Ξ*Ξ,62);
   TITLE
END ↑PREPAREOUTPUT↓;

↑================================================================↓*
```

```
*************************** MAIN PROGRAM ***************************
BEGIN
   * INITIALIZATION *
   ALPHA :=[=,=,=A=,=B=,=C=,=D=,=E=,=F=,=G=,=H=,=I=,=J=,=K=,=L=,=M=,=N=,*
            =O=,=P=,=Q=,=R=,=S=,=T=,=U=,=V=,=W=,=X=,=Y=,=Z=,
            =*=,=>=,=-=,=$=];
   DIGITS:=[=0=,=1=,=2=,=3=,=4=,=5=,=6=,=7=,=8=,=9=];
   CSET :=[15, 17, 18, 19, 20, 21, 22, 23, 26, 35, 36 ];
   LITCHARS :=[=A=,=B=,=D=,=F=,=L=];
   PARAMETER :=[=A=,=B=,=C=,=F=,=L=,=N=,=O=,=R=,=T=,=X=,=Z=,=,=];
   PSEUDOSUBSET :=[ (, 5, 6, 7, 8, 9, 10, 11, 12, 15, 18, 19, 20,
                    21, 22, 23 ];
   NONFATAL :=[0, 1, 28, 30, 40, 41, 42, 43, 45];
   NPROG:=0; WRITE(= =); ENDINPUT:=FALSE;

1: * READ USER'S CARD / CHECK EOF *
   READARRAY(INTEXT); IF ENDINPUT THEN GOTO 3;

   * NEW PROGRAM *
   NPROG:=NPROG+1;

   *RE-INITIALIZE VARIABLES*
   REWRITE(S); REWRITE(R); REWRITE(9);
   ABSPROG:=FALSE; ERRPAGE:=[ ]; LSTFLAG:=TRUE; FATAL:=FALSE; EP:=0;
   TBLPRINT:=FALSE; LSTOUTPUT:=FALSE; PUNCHBIN:=FALSE; LISTBIN:=FALSE;
   AF:=FALSE; RF:=FALSE; OPTION:=" ";
   PG:=0; LI:=0; LN:=1; SN:=-1; ER:=0; NER:=0; LC:=0; SCAN1:=TRUE;

   FOR I:=0 TO SZ DO SYMTAB[I].NAME:=EMPTY;
   FOR I:=1 TO HLT DO LITTAB[I,1]:=NOLIT;

   *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*.LN:=LN; S*.OPFLD:=0; S*.LOCFLD:=0;PUT(S);*
   PRINTARRAY(INTEXT); LINES(2);
   COL:=1; GETNAME(A);
   IF A<>ASMBE THEN ERROR(18);
   REPEAT
      IF CH IN PARAMETER THEN
      CASE CH OF
           =A= : BEGIN ABSPROG:=TRUE; AF:=TRUE; LC:=20000 END;
           =R= : BEGIN ABSPROG:=FALSE; RF:=TRUE; LC:=0 END;
           =L= : LSTOUTPUT:=TRUE;
           =B= : PUNCHBIN:=TRUE;
           =T= : TBLPRINT:=TRUE;
           =N= : OPTION:=CH;
           =Z= : OPTION:=CH;
           =O= : LISTBIN:=TRUE;
   =X=,=F=,=C= : BEGIN WRITE(=PARAMETER C IS IGNORED=); LINES(1) END;
           =,= :
      END ELSE ERROR(19);
      NEXTCHAR(CH);
   UNTIL BLANKOREOL(CH);
```

```
   LSTFLAG:=LSTOUTPUT; BINOUTPUT:=PUNCHBIN OR LISTBIN;
   IF AF=0F THEN
   BEGIN
      ABSPROG:=FALSE; LC:=0;
      WRITE(=* PROGRAM ASSUMED TO BE RELOCATABLE *=); LINES(2)
   END;

   IF ~(LSTOUTPUT v PUNCHBIN v TBLPRINT v LISTBIN) THEN ERROR(22);

   IF NER<>0 THEN
   BEGIN
      WRITE(=ERRORS IN CONTROL CARD=); LINES(1); MULTIPRINT(=*=,22);
      LINES(2); PRINTEPR; SKIPTOEND; PUNCHCARD(ERRCARD); GOTO 2
   END;

   PASSONE;

   IF TBLPRINT THEN ←PRINT SYMBOL TABLE→
   BEGIN
   TITLE: WRITE(=                SYMBOL TABLE=,        EOL,=0=);
      FOR I:=0 TO SN DO
      BEGIN
         N:=SYMPTR[I];
         WRITE(EMPTY,SYMTAB[N].NAME);
         IF ~ABSPROG THEN OUTCH(SYMTAB[N].TYP); OUTCH(= =);
         OUTOCT(SYMTAB[N].LOC); WRITE(EOL,= =);
      END
   END;

   PASSTWO;

   IF ~(ERRPAGE ≤ [ ]) THEN ←PRINT PAGE NUMBERS WITH ERRORS→
   BEGIN
      WRITE(=ERRORS ON THE FOLLOWING PAGES=); LINES(2);
      FOR I:=1 TO EP DO
         IF I IN ERRPAGE THEN  WRITE(I:5);
      LINES(2);
   END;

   IF FATAL THEN BEGIN
                    PUNCHCARD(ERRCARD);
                    WRITE(=BINARY OUTPUT SUPPRESSED=,EOL,= =); GOTO 2
                 END;

   IF LISTBIN THEN TITLE;

   IF BINOUTPUT THEN ←BINARY OUTPUT REQUESTED→
   IF ABSPROG THEN ABSBINRECORDS
   ELSE
   BEGIN ←PUNCH/PRINT IN RELOCATABLE BINARY FORMAT→
      NAMRECORD;
      IF NT>0 THEN ENTRECORD;
      IF XT>0 THEN EXTRECORD;
      IF BPC>0 THEN DBLRECORDS(B,0);
      IF ENDLC>0 THEN DBLRECORDS(R,1);
      ENDRECORD;
   END;

2: IF ~ENDINPUT THEN ←ASSEMBLE NEXT PROGRAM→
                 BEGIN WRITE(EOL,=1=); GOTO 1 END;

3: LINES(2); WRITE(=END OF RUN=); WRITE(EOL);

END.
```

APPENDIX H

SAMPLE RUNS UNDER THE CROSS ASSEMBLER



TITLE:

EXAMPLE OF A PROGRAM WITHOUT ERRORS

PROGRAMMER:

V. ANDREEV

PAGE 1 EXAMPLE OF A PROGRAM WITHOUT ERRORS

ASMB,L,T,G

* PROGRAM ASSUMED TO BE RELOCATABLE *

** NO ERRORS DETECTED IN PASS 1 **

PAGE 2 EXAMPLE OF A PROGRAM WITHOUT ERRORS

SYMBOL TABLE

START	R	000000
.FAC	X	000001
SL	R	000035
.ACA	R	000054
INPLT	R	000055
CLTM	R	000056
[illegible]	R	[illegible]
CTR	R	000156
.CTC.	X	000006
.WAR.	X	000003
.ICH.	X	000004
.DTA.	X	000005
CLTPT	R	000157

PAGE 3 EXAMPLE OF A PROGRAM WITHOUT ERRORS

```
 1                                ASMB,L,T,C
 2   000000                             NAM ITTAL
 3   000000  000000         START NOP
 4   000001  062162  R                  LDA =D-31
 5   000002  072155  R                  STA CTR
 6   000003  062163  R                  LDA =D5
 7   000004  006404                     CLB,INB
 8   000005  016002  X                  JSB .DIO.
 9   000006  000005                     ABS 5
10   000007  000014  R                  DEF *+5
11   000010  016004  X                  JSB .IOR.
12   000011  104400                     DST INPUT,I
     000012  100055  R
13   000013  016003  X                  JSB .RAR.
14   000014  066055  R                  LDB INPUT
15   000015  046164  R                  ADB =D2
16   000016  076055  R                  STB INPUT
17   000017  036155  R                  ISZ CTR
18   000020  026003  R                  JMP START+3
19   000021  062162  R                  LDA =D-31
20   000022  072155  R                  STA CTR
21   000023  104200                     DLD =F0.0
     000024  000165  R
22   000025  1044[illegible]            DST ANS
     000026  000164  R

24   [illegible]
25   [illegible]
26   [illegible]                        DST ANS
27   [illegible]            S[illegible]  LDA [illegible]
28   [illegible]                        ADA =D2
29   [illegible]  036155  R             ISZ CTR
30   [illegible]  076054  R             STA [illegible]
31   000041  026035  R                  JMP S[illegible]

33   000042  062164  R                  LDA =D2
34   000043  002400                     CLB
35   000044  016002  X                  JSB .DIO.
36   000045  000157  R                  DEF OUTPT
37   000046  000053  R                  DEF *+5
38   000047  104200                     DLD ANS
     000050  000164  R
39   000051  016004  X                  JSB .IOR.
40   000052  016005  X                  JSB .DTA.
41   000053  102077                     HLT 77B

43   000054  000056  R       .0[illegible]  DEF MONTH
```

PAGE 4 EXAMPLE OF A PROGRAM WITHOUT ERRORS

```
44  000055 000056 R     INPUT DEF MONTH
45  000056 000000       MONTH BSS 6
46  000154 000000       A.S.  BSS 2
47  000156 000000       CTR   BSS 1
48                            EXT .IC...PA...IOR...CTA.
49  000157 024106       OUTPT ASC 3,(FR.2)
    000160 034036
    000161 031051
    000162 177741
    000163 000003
    000164 000032
    000165 000000
    000166 000000
50                          1   END START
```

** NO ERRORS DETECTED IN PASS 2 **

PAGE 5 EXAMPLE OF A PROGRAM WITHOUT ERRORS

NAM RECORD

[illegible]	[illegible]	012447	052117	052101	046040	006167	000000
000[illegible]	000000	000000	000000	000000	000000	000000	000000
000[illegible]							

EXT RECORD

[illegible]	[illegible]	127366	[illegible]	044544	020[illegible]	027104	044517
027[illegible]	027122	041522	027[illegible]3	027111	047522	027004	027104
052101	[illegible]						

DEL RECORD

[illegible]	[illegible]	134376	[illegible]	013320	000000	062000	000162
072[illegible]	[illegible]	062[illegible]	[illegible]	006406	100300	016002	000000
[illegible]	[illegible]	1[illegible]	[illegible]	100[illegible]55	016013	066000	000055
[illegible]	[illegible]	[illegible]	[illegible]	133320	036000	000156	026000
[illegible]	[illegible]	[illegible]	[illegible]	000156	114200	020202	000166
[illegible]	[illegible]	[illegible]	[illegible]	102[illegible]	014001	000154	104400
[illegible]	[illegible]	[illegible]	[illegible]	04[illegible]	000164	[illegible]	000166
[illegible]	[illegible]	[illegible]	[illegible]				

DEL

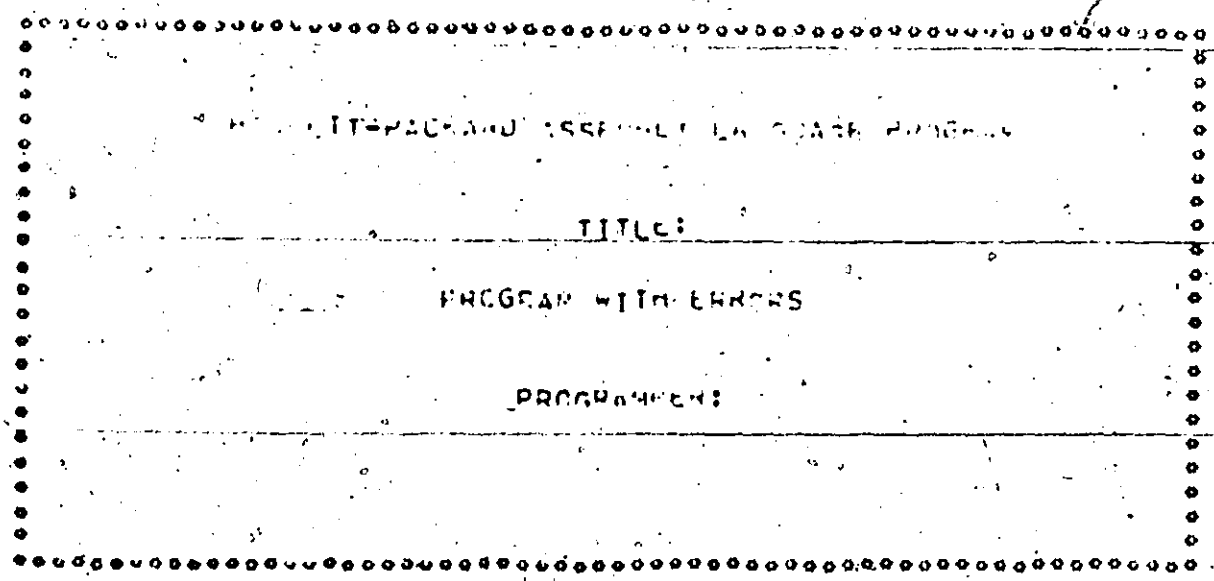
[illegible]	[illegible]	[illegible]	[illegible]	121[illegible]	062[illegible]	000164	006400
[illegible]	[illegible]	[illegible]	[illegible]	104[illegible]	000154	016004	016005
102[illegible]	[illegible]	[illegible]	[illegible]				

DEL RECORD

007[illegible]	[illegible]	[illegible]	[illegible]	000[illegible]	024106	034056	031051
17774[illegible]	[illegible]	[illegible]	000002	000[illegible]	000000		

END RECORD

002[illegible]	1[illegible]	120001	000000


TITLE:
PROGRAM WITH ERRORS
PROGRAMMER:

```
PAGE      1                         PROGRAM WITH ERRORS

ASMB,R,L,T,Z


ERRORS DETECTED IN PASS 1
*************************

(ERROR MESSAGES ARE LISTED BELOW THE ERRONEOUS LINES)

     52             ASC  31,
***** ERROR: ERROR IN PSEUDO-OP ASC: <1 OR >28

     68             IFN
***** ERROR: AN IFN OR AN IFZ FOLLOWS EITHER AN IFN OR AN IFZ
             WITHOUT AN INTERVENING XIF. THE SECOND PSEUDO
             INSTRUCTION IS IGNORED

     78             JMP  C,I
***** ERROR: ILLEGAL OPERATION

    117  [illegible]
***** ERROR: SYMBOL TOO LONG, TRUNCATED TO 5 CHARACTERS

    117             [illegible]
***** [illegible]
***** [illegible]


.DIC.
***** ERROR: ABOVE ENT SYMBOL APPEARS IN A COM OR AN EXT STATEMENT

ERR
***** ERROR: ABOVE ENT SYMBOL IS NOT A LABEL OF ANY INSTRUCTION

LAB-1
***** ERROR: ABOVE ENT SYMBOL IS NOT A LABEL OF ANY INSTRUCTION


**     9 ERRORS DETECTED IN PASS 1 **
```

1

PAGE 2 PROGRAM WITH ERRORS

SYMBOL TABLE

START	R	000000
.ICC.	A	000001
FLOAT	X	000002
IFIX	X	000003
SQRT	X	000004
.CIC.	A	000005
.FCI.	A	000006
.LTA.	X	000007
.[illegible].	A	000010
.ICH.	X	000011
.[illegible].	X	000012
.[illegible]	R	000002
.[illegible]	R	000003
[illegible]	R	000004
[illegible]	R	000010
[illegible]2	R	000013
[illegible]3	R	000023
[illegible]	R	000044
.F[illegible]	R	000013
[illegible]A	R	000137
45[illegible]	X	000014

PAGE 3 PROGRAM WITH ERRORS

1 ASM.R.L.T.

PAGE 4 LINE FORMULI : DISTANCE, SLOPE, MID-POINT

```
                                *
                                *
000000                          NAM LINE
000000 000000           START   ZCP
000001 024020                   JMP INPUT
***** ERROR: UNDECLARED VARIABLE
                                EXT .CIO.,ERR,START
                                EXT .IOC.,FLOAT,FIX,SQRT
                                EXT .IC.,.IOI.,.DIA.,.ERR.
                                EXT .IOB.,.IM.
000012 000034 R         .DATA   DEF DATA
000013 000044 R         .PRIN   DEF PRINT
000014 000000           DATA    BSS 4
000015 024100           FMT     ASC 3,(F8.3)
000016 034456
000017 031441
000020 024100           FMT2    ASC 8,(F8.3,2X,2X,F8.3/)
000021 034456
000022 031456
000023 023456
000024 023456
000025 024120
000026 027463
000027 027441
                                SUB
000030 024100           FMT3    ASC 3,(4X,)
                                END
```

```
21                                  * INPUT THE FIRST TWO POINTS: FOUR DATA
22   000026 000000                    NOP
23   000027 062167 R                  LDA =D5
24   000030 006404                    CLB,INB
25   000031 016025 X                  JSB .DIO.
26   000032 000023 R                  DEF F113
27   000033 000037 R                  DEF *+4
28   000034 062170 R                  LDA =D4
29   000035 024000                    JMP ASDF
***** ERROR: UNDECLARED VARIABLE
30   000036 066072 R                  LDB .DATA
31   000037 016012 X                  JSB .TAR.

33                                  * THE DISTANCE BETWEEN THE TWO POINTS
34                                    1,2
35   000040 062006 R                  LDA DATA+2     LOAD X2 INTO REGISTER A
36   000041 003004                    CMA,INA        NEGATE A (I.E. A=-A)
37   000042 042004 R                  ADA DATA       ADD X1 TO A (A=X1-A)
39   000043 126056 R                  JMP *+5
40                         PRINT      * P 4
41   [illegible]
     [illegible]
     [illegible]
     [illegible]
43                                    S P
     000051 072044                    STA PRINT
     000052 072044 R                  STA PRINT
     000054 072044 R                  STA PRINT
44                                    SZB
45   000051 100200                    MPY PRINT
     000053 100200                    MPY PRINT
     000055 100200                    MPY PRINT
     000057 100200                    MPY PRINT
46   000053 062000 R                  LDA =D22
***** ERROR: INVALID LITERAL
47   000054 072044 R                  STA PRINT
49   000055 062007 R                  LDA DATA+3
50   000056 003004                    CMA,INA
51   000057 042005 R                  ADA DATA+1
52                                    ASC 1
***** ERROR: ERROR IN RSEL... ASCII <1 OR >28
53   000060 072045 R                  STA PRINT+1
54   000061 100201                    MPY PRINT+1
55   000063 042044 R                  ADA PRINT
57   000064 066000 R                  LDA =F1.5
***** ERROR: LITERAL INCOMPATIBLE WITH THE INSTRUCTION USED
```

PAGE 0 LINE FORMULI : DISTANCE, SLOPE, MID-POINT

```
                                JSR FLOAT
                                JSR SQRT
                                DST POINT
                                XIF


                        *   FIND THE SLOPE OF THE LINE
                                IEM
                                LDA DATA+2
                                CMP.I A
                                ADA DATA
                                IFC.
                                JMP *+5
                        POINT MEM
                                NOP             ;NO-OPERATION
                                STA POINT
                                SEC I
                                LDA DATA+3
                                ADA 7777777B
                                DIA 7777B
                                CMP.I A
                                FAD DATA  *1  ADD 1 TO M
                                CLA             CLEAR A
                                LAV POINT
                                DST POINT
                                XIF


                        *   OUTPUT THE RESULT
                                LDX = 2
                                CLA
                                JSR *.IO.
                                DFF FIX
                                DFF *+4
                                FLD POINT
                                JSR *.IOB.
                                JSR *.IA.


                        *   FIND THE MIDPOINT OF THE LINE SEGMENT
                                LDA DATA
                                ALA DATA+2
                                CLM
                                MUL OUT
                                JSR FLOAT
                                FMP =F.5
                                DST POINT

                                LDA DATA+1
                                ALA DATA+3
                                CLM
```

```
000115 016002 X            JSB FLOAT
000116 016013 X            FMP =F.5
000120 104400              DST POINT+2

000122 062171 R            LDA =+2
000123 006400              CLB
000124 066000 R            LDB =+32
***** ERROR: INVALID LITERAL

000125 016005 X            JSB .TIC.
000126 000013 R            DEF FLT2
000127 000253              OCT 253,358,64275965,293
***** ERROR: ILLEGAL CHARACTER IN AN OCTAL CONSTANT

***** ERROR: ILLEGAL CHARACTER IN OPERAND FIELD
***** ERROR: ILLEGAL CHARACTER IN AN OCTAL CONSTANT

***** ERROR: ILLEGAL CHARACTER IN OPERAND FIELD

000133 000100              DEC 64,654,2345789974,172
***** ERROR: ILLEGAL CHARACTER IN OPERAND FIELD

***** ERROR: CONSTANT GREATER THAN 177777B

***** ERROR: ILLEGAL CHARACTER IN OPERAND FIELD
***** ERROR: ILLEGAL CHARACTER IN [illegible]

[illegible]
***** [illegible]

000140 000145              DEF [illegible]
                           ASC [illegible]
***** [illegible] IS [illegible] FIVE [illegible] OTHER THAN [illegible]
***** ERROR: ERROR IN PSEUDO-OP ASC: [illegible]

000141 062171 R            LDA =+2
000142 066003 R            LDB .PRIT
000143 016010 X            JSB .[illegible]
000144 016007 X            JSB .[illegible]
                           [illegible]

                           EXT ASD
000145 026000 X            JMP ASD+1
***** ERROR: EXTERNAL SYMBOL SHOULD APPEAR ALONE IN OPERAND

000146 026000              JMP INPUT
***** ERROR: UNDECLARED VARIABLE

000147 001070              ALS,CLE,SLA,ALS
000150 001040              ALS,CLE
000151 000070              CLE,SLA,ALS
000152 000050              CLE,SLA
000153 004050              CLE,SLB
```

```
PAGE    4          LINE FORMULI : DISTANCE, SLOPE, MID-POINT

  134  000154 001020          ALS,ALS
  135  000155 002571          CLA,SEZ,CLE,SSA,SLA,RSS
  136  000156 002051          SEZ,SLA,RSS
  137  000157 006500          CLA,CLE
  138  000160 002114          CLE,SLA,INA
  139  000161 003004          CLA,INA
  140  000162 000003          ALS,CLE,SLA
***** ERROR: MIXING OF A, B TYPE INSTRUCTIONS IS ILLEGAL

  141  000163 000000          ALS,SLA,CLE
***** ERROR: ILLEGAL SEQUENCE OF OPS IN A MULTIPLE INSTRUCTION

  142  000164 000000          ALS,MCP,CLE
***** ERROR: ILLEGAL OP IN MULTIPLE INSTRUCTION

  143  000165 000000          CLA,SSA,SEZ,RSS
***** ERROR: ILLEGAL SEQUENCE OF OPS IN A MULTIPLE INSTRUCTION

  144  000166 000000          CLA,SSB,*+2
***** ERROR: MIXING OF A, B TYPE INSTRUCTIONS IS ILLEGAL

  145                         END

** 24 ERRORS DETECTED IN PASS 2 **

ERRORS PRINTED IN PASS 2
    1    4    5    7    8

BINARY OUTPUT SUPPRESSED
```

REFERENCES

1. Gear, C.W. - 'Computer Organization and Programming' McGraw Hill, New York (1969).

2. High Level Languages - International Conference on State of the Art Report - INFOTECH Information, Berkshire, England.

3. C.A.R. Hoare and N. Wirth, "An axiomatic definition of the programming language PASCAL", Berichte der Fachgruppe Computer-Wissenschaften, ETH, No.6 (Dec. 1972)

4. Hewlett-Packard 2100A Reference Manual, Hewlett-Packard Corp. 11000 Wolfe Road, Cupertino, Calif. (HP 02100-90001).

5. Hewlett-Packard Assembler Manual, Hewlett-Packard Corp. 11000 Wolfe Road, Cupertino, Calif.(HP 02116-9014).

6. Gries, D., Compiler Construction for Digital Computers, J. Wiley & Sons Inc. (1971).

7. Basic Control System, Hewlett-Packard, Hewlett-Packard Corp. 11000 Wolfe Road, Cupertino, Calif. (HP 02116-9017).

8. Control Data 6000 Computer System, SCOPE Reference Manual Publ.No. 60327300, pg. 4-32.