Cross Assembly and Relocation of Programs for the Intel Microprocessors using a PDP-15 as Host Computer

Andersson, Leif

1976

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Andersson, L. (1976). *Cross Assembly and Relocation of Programs for the Intel Microprocessors using a PDP-15 as Host Computer*. (Research Reports TFRT-3098). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

CROSS ASSEMBLY AND RELOCATION OF PROGRAMS FOR THE INTEL MICROPROCESSORS USING A PDP-15 AS HOST COMPUTER

L. ANDERSSON

Report 7602 May 1976
Department of Automatic Control
Lund Institute of Technology

CROSS ASSEMBLY AND RELOCATION OF PROGRAMS FOR THE INTEL MICROPROCESSORS USING A PDP-15 AS HOST COMPUTER

L Andersson

ABSTRACT

In the preparation of programs for small computers it is an advantage to use a larger computer as a host computer. The larger computer's powerful text editor, mass storage etc can be utilized with a considerable gain in programmer's time and comfort. The paper describes a project where a PDP-15 has been used as a host computer with the Intel microprocessors as the target computers. The PDP-15 assembler produces the code for the microcomputers, which can be achieved without modifying the assembler as such. The target machine instructions need simply to be defined in a separate source file. The output of the assembler is a relocatable binary file. This is further processed in a program package which relocates and links the program units. The relocating program also performs a library search, and this is considered an essential feature of the programming tool. It is highly modular in structure, and this facilitates extensions to other target machines.

1. INTRODUCTION

There are two commercially available programming tools for microprocessors. The first one consists of prototyping systems like the INTELLEC with software support such as text editors and assemblers. The second one consists of cross assemblers and simulators on large time-sharing computers.

The first method has the disadvantage that it is in general completely paper tape oriented. This makes the task of error correction and reassembly very slow and tedious. The second method gives access to the host computer's powerful text editor, mass storage etc which is of great value. However, none of the two methods in general has facilities for relocation and linking of binary object code modules or for selective search in program libraries.

The report describes a programming tool which includes these facilities. The host computer is a PDP-15 and target computers are the INTEL family of microcomputers: 4004, 4040, 8008 and 8080.

The host computer's assembler produces relocatable code for the microprocessors. This is further processed by program packages called LOAD4 for the 4004/4040 and LOAD8 for the 8008/8080. These packages have the following features:

- Named program files are relocated.
- The start address is given only for the first program.
- Subroutine calls and entry points are linked automatically.
- A library file may be specified.
- This library file is selectively loaded, i.e. only those program units which are requested by previously processed programs appear in the output.

The report consists of two parts. Part I is a user's manual for the assembler and for LOAD4/LOAD8 and Part II describes the implementation in detail. The appendices contain instruction summaries, macro file listings and program listings.

The programming tool was originally developed for the 8008 and 8080. The modifications necessary for the 4040 were made as a master's thesis by J. Miszsuk and P. Wolgast [4].

2. ASSEMBLER

It is often quite simple to make an assembler produce code for another machine than it was originally written for. The task of an assembler is, broadly speaking, to translate symbols such as instruction mnemonics or symbolic addresses into numeric values, i.e. machine instructions or numeric addresses. If the instruction mnemonics of the target computer are entered into the assembler's symbol tables, the assembler will recognize them and produce the correct object code. Simple instructions may be defined by direct assignment statements, while more complicated ones are handled using macros.

The advantages of using this method rather than writing a dedicated assembler are:

- It takes much less time, typically a few man-days for simple processors like the Intel family.

- It is more powerful. All the macrofacilities and other features of the original assembler may be used by the microcomputer programmer.

The disadvantages are:

- It does not work on all host computers. The crucial point is the order in which the assembler scans its symbol tables. If it scans the user's symbol table before its own table of instruction mnemonics, the scheme will work, otherwise the choice of target machine mnemonics will be severely restricted.

- The error checking is incomplete. For example, a symbol which is an instruction mnemonic for the host computer but not for the target computer would of course not be flagged.

- The output of the assembler must be further processed. Even if the assembler were to produce absolute code, the format would probably not be consistent with the requirements of a PROM programmer.

The definitions of the mnemonics for the 4040, 8008, and 8080 are collected in files called ASS40, ASS8 and ASS80 respectively. The appropriate macro file is then scanned by the assembler each time a new source program is to be assembled.

The resulting assemblers for the different microprocessors are quite similar to those provided by Intel for the Intellec or on time-sharing systems. There are minor differences in the use of tabs, spaces etc, and the pseudo ops are, of course, different.

The following sections constitute a User's Manual for this assembler. It is reasonably self-contained, although a general familiarity with assembly programming and especially the MACRO-15 assembler is of value.

2.1. Instruction Mnemonics.

The assembler will recognize the instruction mnemonics as given in the various user's manuals. Appendices A - C contain summaries of the instruction sets. This section contains some clarifications and also some extensions.

2.1.1. 4004 and 4040.

The 4004 instruction set is a true subset of the 4040 set and therefore only one macro file is necessary. The registers of the 4040 are referred to as R0, R1, R2 --- R8, R9, R10 --- R15. See Appendix A.

INC, ADD, SUB, LD and XCH are one word instructions where a register is indicated. The form of the instruction in the source program is as in the following example:

```
ADD R3    /ADD REG. R3 TO AC
```

SRC, FIN and JIN also specify a register. However, in this case only the even-numbered registers are legal, i.e. R0, R2, R4 --- R14.
NOTE! No check is made on the legality of the register number.

FIM specifies a register pair and data to be loaded into that register pair. The form is:

```
FIM R4, DATA
```

Only even-numbered registers are legal. DATA is an eight-bit number or a symbol having a value in that range.

JUN and JMS have the form

```
JUN LABEL
```

where LABEL is defined either in a label part of a statement or in a .GLOBL pseudo-op (see sects. 2.2 or 2.6).

The JCN instruction, Jump Conditionally, has been split up into several mnemonics as follows:

JTT   Jump if True Test
JTC   Jump if True Carry
JTZ   Jump if True Zero, i.e. AC = 0
JTTC Jump if True Test or Carry
JTTZ Jump if True Test or Zero
JTCZ Jump if True Carry or Zero
JTTCZ Jump if True Test, Carry or Zero

JFT Jump if False Test
JFC Jump if False Carry
JFZ Jump if False Zero, i.e. if AC ≠ 0
JFTC Jump if False Test and Carry
JFTZ Jump if False Test and Zero
JFCZ Jump if False Carry and Zero
JFTCZ Jump if False Test, Carry and Zero

The form of the instruction is

```
JTZ LABEL
```

ISZ has the form

```
ISZ R3, LABEL
```

BBL and LDM have the form

```
BBL DATA
```

where DATA is a four-bit number or symbol.

2.1.2. 8008.

All mnemonics in the MCS-8 user's manual except INP and OUT are recognized by the assembler. See Appendix B and [2].

JMP and CAL instructions have the form

```
JMP LABEL
```

where LABEL is defined in a label part of a statement or in a .GLOBL pseudo op. The jump-instructions are three word instructions, and the assembler and loader together will take care of the splitting into three words.

The immediate-instructions, Load Immediate, Add Immediate etc are two-word instructions of the form

```
ADI DATA
```

where DATA is an eight-bit number or a symbol with a value in that range.

The Restart-instruction, RST, has the form

```
RST n
```

where n is any of the numbers 00, 10, 20, 30, 40, 50, 60, 70, or a symbol having any of these values.

2.1.3. 8080.

The register names recognized by the assembler are: A, B, C, D, E, H, L, M, SP and PSW. M is the memory cell addressed by the H, L registers. SP is the stack pointer and PSW is accumulator and flags. See Appendix C and [3].

MOV-instruction.
The form is

```
MOV A,B
```

where all register names except SP and PSW are legal.

MVI.
The form is

```
MVI B, DATA
```

where DATA is an eight-bit number.

The other Immediate instructions have the form

ADI DATA

where DATA is an eight-bit number.

JMP, CALL and Restart instructions are the same as for the 8008.

LXI-instruction.
The form is

LXI B, DATA

where B is the register name (legal B, D, H, SP) and DATA is a 16-bit number. The number is loaded with the most significant half in B and least significant part in C.

STA, LDA, SHLD, LHLD.
The form of these instructions is

SHLD BANK, CELL

where BANK and CELL are eight-bit numbers or symbols with values of that range.

2.2. Statement Format.

Statements are written one per physical line and terminated by a carriage return. A statement may contain up to four fields, the label field, the operation field, the address field and the comment field, separated by a space or a tab, in the following denoted ⇥.

LABEL ⇥ OPERATION ⇥ OPERAND ⇥ /COMMENT

The label field may be empty or contain a symbolic label. When a label is encountered, the symbol is said to be defined. A label symbol cannot be redefined. Even if the label field is empty, the delimiting tab or space must be present.

The operation field may contain an instruction mnemonic, a pseudo-op code, a macro name, a number or a symbol which is not a label.

The operand field may be empty or contain one or two operands separated by a comma. An operand may consist of a symbol, a numeric constant or an expression.

The comment field may be empty or contain any text preceded by a slash (/). A comment may appear alone in a line, in which case it may be preceded by any number of tabs or spaces, including zero.

2.3. Symbols.

The programmer creates symbols to represent addresses, operation codes, numeric values and macros. A symbol contains from one to six characters from the following set:

The letters A - Z
The digits 0 - 9
The special characters . and %

The first character of a symbol must not be a digit. A symbol declared in a .GLOBL statement (see sect. 2.6) must not contain . or % because LOAD4 and LOAD 8 cannot handle these.

A symbol is defined and given a value by

1. Its appearance as a label.
2. A macro definition (sect. 2.6).
3. A .GLOBL statement.
4. A direct assignment statement.

2.4. Direct Assignment Statements.

The programmer may define a symbol by means of a direct assignment statement of the form:

```
SYMBOL = n
```

or

```
SYM1 = SYM2
```

where n is any number or expression. The statement must begin in the first position of a line, i.e. no leading tabs or spaces, and must not contain any tabs or spaces.

The main use of direct assignment statements in programs for the Intel microcomputers is to create mnemonic names for data addresses. Program and data are often in separate parts of the memory since programs normally reside in ROM and data in RAM. It is then convenient to give the address of the data a symbolic name so that the addresses may be referenced by their names and a possible change need be done only in one place, and also to improve readability of the source text.

2.5. Expressions.

Expressions are strings of symbols and numbers separated by arithmetic and Boolean operators. Expressions represent unsigned numeric values. The following are legal operators to be used with expressions:

Symbol:	Function:
+	Addition (two's complement)
-	Subtraction (two's complement)
*	Multiplication (unsigned)
/	Division (unsigned)
&	Logical AND
!	Inclusive OR
\	Exclusive OR
,	Exclusive OR

Operations are performed from left to right, i.e. no operator precedence and no parentheses.

Example of direct assignment and expression:

```
VAR = 015
        LLI VAR+3
```

2.6. Pseudo Operations.

As mentioned earlier, the symbol table contains definitions of symbols which are instruction mnemonics, macro names etc. A class of symbols called pseudo operations is also contained in the symbol table. Instead of generating instructions these symbols tell the assembler how to proceed with the assembly process. A subset of these pseudo instructions are relevant also to the micro computer, programmer. This section contains a summary and also a more extensive description of some of them. For further information see the MACRO-15 manual [5].

.DEC	Sets prevailing number radix to decimal.
.DEFIN	Defines macros.
.EJECT	Skip to top of form on listing device.
.END	Must terminate every source program.
.ENDC	Terminates conditional coding
.ENDM	Terminates a macro definition.
.ETC	Used in macro definitions to continue the list of dummy arguments on succeeding line.
.GLOBL	Used to declare all internal and external symbols which reference or are referenced by other programs.
.IFxxx	Conditional assembly.
.LST	Source lines between .NOLST and
.NOLST	.LST are not listed.
.OCT	Sets the prevailing number radix to octal. Default value at start.
.TITLE	The character string after .TITLE is printed on top of each listing page.

2.6.1. Radix control (.OCT and .DEC).

The initial radix (base) used in all number interpretation by the assembler is octal. This may be changed to decimal and back using the pseudo-ops .DEC and .OCT. All numbers are decoded in the current radix until a new radix control pseudo op is encountered. The programmer may change radix at any point in the program.

2.6.2. Macro definition (.DEFIN, .ETC and .ENDM).

In its simplest form a macro is an abbreviation for a sequence of instructions. The macro processor part of the MACRO-15 assembler is, however, very powerful, and a description would be rather lengthy. The reader is therefore referred to [5] chapter 4. The Intel programmer may take full advantage of the macro facilities subject to the following limitations:

1. The generated source lines must naturally be legal and meaningful, 4040, 8008 or 8080 code.

2. Since macro nesting, i.e. macro calls within macro calls, is limited to three levels by MACRO-15, and since many of the micro computer op codes are defined as macros, the practical nesting depth is two levels.

2.6.3. Global and external symbols (.GLOBL).

Relocatable programs often refer to symbols, normally subroutine entry points, in separately assembled program units. Such symbols are called external symbols. In the called program the same symbol is called a global symbol. Both external and global symbols must be declared so that the assembler can pass information enabling the loader to link the two together.

They are both declared by the pseudo-op .GLOBL

Ex.: .GLOBL SUB1, SUB2, SUB3

2.7. Operating Procedure.

The assembler is called by typing MACRO after the monitor's $ request. The assembler identifies itself by typing

```
MACRO-15 Vnn
>
```

and the user can type his command.

The command string format consists of a string of options followed by a left arrow followed by the name of the definition file and the program file

options←file name 1, file name 2

Ex.: BNFQ←ASS8,PROG

The options given in this example are those which are normally used when assembling programs for the Intel micro computers. The meaning of the options is as follows:

Opt.	Action
B	Generate a binary file output.
F	Read a macro definition file before the program file. In this case it is the file ASS8, which contains the definitions of Intel 8008 instructions.
N	Generate a program listing with each line numbered.
Q	Do not print the generated source lines of macro expansion.

The Q-option is a modification of the normal MACRO-15 assembler. See sect. 4.3.

Other options are possible, and the reader is referred to [5] for a complete list.

3. LOAD4 and LOAD8.

LOAD4 and LOAD8 relocates and links binary programs produced by the assembler for the Intel micro computers, LOAD4 applies to the 4004/4040 and LOAD8 to the 8008/8080. In the following only LOAD8 is mentioned, but everything applies to LOAD4 as well.

The operator specifies the start address of the first program, the names of the program files and optionally a library file. Both the library file and the program file may consist of concatenated program units. (A program unit is the result of the assembly of one source program. A program file contains one or more program units.) The concatenation may be done using either the PIP or the UPDATE utility program. The difference between a program file and a library file is that all the program units of a program file are loaded, while only those units of a library file are loaded which are requested by previously loaded program units.

3.1. Command String Syntax and Semantics.

LOAD8 recognizes four classes of commands: LIBR, GLOBL, LOAD and STOP. The syntax of these commands is specified using a modified Backus-Naur notation [6] with the additional symbols [<item>] denoting that the item within the brackets is optional, and <item>... denoting one or more repetition of <item>.

3.1.1. Common items.

Certain basic items are not defined here in which case they are considered self evident or the definition is identical to the definition in the Algol report [6].

When nothing else is stated items are separated by one or more spaces. The symbol ↲ denotes carriage return and ⊠ denotes altmode.

<file name>::=<identifier, max 5 characters>
<global name>::=<identifier, max 6 characters>
<address>::=<bank number>:<address within bank>
<bank number>::=<octal number, range implementation dependent>
<address within bank>::=<octal number, range 0-377>
<delimiter>::=,|<space>|↲|<delimiter>...

3.1.2. LIBR command.

Syntax:

<LIBR command>::=LIBR<file name>↲

Ex.: LIBR LIB8 ↲

File name specifies the name of the library file. The default assumption is NONE which indicates that no library is to be used.

3.1.3. GLOBL command.

Syntax:

<GLOBL command>::=GLOBL<global name><address>
[<delimiter><global name><address>]...
[<delimiter>]⊠

Ex: GLOBL SUB1 10:037, SUB2 11:122⊠
GLOBL GLOB1 0:165
GLOB2 01:15⊠

The GLOBL command gives LOAD8 the final address of a global symbol before scanning of the program files and the library file starts. This is desirable on two occasions:

1. When the global symbol does not exist in any of the files.

2. When the program unit containing the symbol has already been loaded. If this unit exists in the library, loading of another copy is prevented by giving the global names in a GLOBL command.

In case 1 above it is not strictly necessary to give the globals in a GLOBL command, because LOAD8 will ask for the final addresses of any unresolved globals.

3.1.4. LOAD command.

Syntax:

<LOAD command>::=LOAD[<file name>]<address>
<file name>[<delimiter><file name>]...
[<delimiter>]⊠

```
Ex 1: LOAD DFIL 010:312 PROG,SUB1
      SUB2, SUB3 ⊠

Ex 2: LOAD 010:312 PROG,SUB1
      SUB2  SUB3 ⊠
```

The LOAD command is the command which actually starts the relocating procedure. The examples show two forms where the difference is the file name appearing between LOAD and the address. The presence of this file name indicates that the output is to be on disk with the file name as given in the command and extension ABS. If no file name is to the paper tape punch.

3.1.5. STOP command.

Syntax:

<STOP command>::=STOP ↵

This command terminates LOAD8.

3.2. Operating Procedure.

The operator starts LOAD 8 by typing E LOAD8 at the monitor's $ request. The program identifies itself:

```
LOAD8 Vnn
>
```

and the operator can type his commands.

The LIBR and GLOBL commands can appear in any order, and the GLOBL command can appear any number of times. If more than one

GLOBL command appears, the list of global symbols defined by subsequent commands is simply appended to the list given by the previous commands.

If any program contains external references which cannot be resolved, LOAD8 will ask for the addresses as in the following example:

```
UNRESOLVED GLOBALS
SYM1  >
```

The operator should answer with an address in the same format as in the GLOBL command.

3.3. Output from LOAD8.

The output from LOAD8 goes to three units: the line printer, the paper tape punch and the disk.

The line printer output consists of:

1. The program file names.
2. The program unit names with start addresses.
3. The global symbols with addresses.
4. The relocated code in octal format with address information.

The paper tape output starts with a frame of all ones, then the relocated code in absolute binary form, one word/frame, and last a checksum frame.

The disk output is the relocated code in octal format, which is subsequently to be read by a simulator program.

4. IMPLEMENTATION OF THE ASSEMBLER.

The files ASS40, ASS8 and ASS80 contain the definitions of the mnemonic symbols for the 4040, 8008 and 8080 respectively. This section describes examples of these definitions, and Appendices D - F contain complete listings of the files. For reference purposes a description of the output from the assembler and of the special character representation of the assembler output is also included in this chapter.

4.1. The Definition Files.

The simple one-word instructions, like the rotate instructions for all three machines are defined by direct assignment statements.

Example (4040):

```
RAL = 365
```

One-word instructions which designate registers are defined by macros as follows:

Example (8080):

```
.DEFIN MOV, R1, R2
R1&7 + R2 + 100
.ENDM
```

Example (4040):

```
.DEFIN ADD, R
R&17 + 200
.ENDM
```

The 8080 has some double-word load instructions like the LXI. The macro is:

```
.DEFIN LXI, R, DAT
R&6 + 1
DAT& 377
DAT/400
.ENDM
```

The various forms of jump instructions require special treatment since these are the relocatable instructions. For the 8080 the macro is:

```
.DEFIN JMP, ADR
303
ADR
.
.ENDM
```

This applies also to the 8008 although the instruction code is 104 instead of 303. As can be seen the assembler will put the entire address in the second word. However, the 8008 and 8080 must have the address split up into two words. This is done by LOAD8 after relocation.

Furthermore, the period in the third word means that the assembler will put the unrelocated address of the data word itself in this position. This has been done for two reasons. Firstly, to enable the assembler to give a correct size information, and secondly, the various forms of jump instructions are the only ones that are relocatable. The data word containing its own address will then enable LOAD8 to do some error checking.

The jump instruction for the 4040 has the following macro:

```
.DEFIN JUN, ADR
100
ADR
.ENDM
```

In this case the most significant four bits of the address should be in the first word. The splitting is done by LOAD4 which consequently has to delay its output one step so that the modification can be done.

4.2. Modifications of the MACRO-15 Assembler.

Strictly speaking no modification at all is necessary to make the MACRO-15 assembler accept Intel programs. All information is contained in the macro files. However, due to the fact that many of the instructions are defined as macros, the listing became rather unpleasant. An extra option Q with the meaning: Do not print the generated source lines of macro expansions was therefore incorporated. The difference shows in the following example of a piece of a program listing without and with Q-option.

```
1      00000 R                 LOOP    LXI H,37253
       00000 R 000041 A *G             H&6*10+1
       00001 R 000253 A *G             37253&377
       00002 R 000076 A *G             37253/400
2                                      MOV D,M
       00003 R 000126 A *G             D*10+M+100
3                                      JMP LOOP
       00004 R 000303 A *G             303
       00005 R 000000 R *G             LOOP
       00006 R 000006 R *G             .

1      00000 R                 LOOP    LXI H,37253
       00000 R 000041 A *G
       00001 R 000253 A *G
       00002 R 000076 A *G
2                                      MOV D,M
       00003 R 000126 A *G
3                                      JMP LOOP
       00004 R 000303 A *G
       00005 R 000000 R *G
       00006 R 000006 R *G
```

The modification of MACRO-15 consisted essentially of code to recognize the Q-option and set a switch QSWCH. A carriage return character is inserted into the output line after the binary code in the listing. The carriage return is inserted under the following condition: If it is a macro expansion and QSWCH is on then insert carriage return.

4.3. The Output From MACRO-15.

The output from MACRO-15 consists of a sequence of four-word blocks of the following format

Code 1	Code 2	Code 3
Data word 1		
Data word 2		
Data word 3		

Code 1 is a six-bit number describing data word 1, code 2 describes word 2 etc.

The following codes, given in octal notation, are relevant to LOAD8. (For an extensive description see [7].)

01 Program size.

04 Absolute instruction or constant.

05 Relocatable address.

07 Symbol, first three characters.
The character representation is Radix 50_8, see sect. 4.4.

10 Symbol, last three characters.

11 External symbol definition.
When the assembler encounters an external reference declaration (by the .GLOBL pseudo-op), it sets aside an extra word of storage. Every subsequent reference to the external symbol is then translated into a reference to this extra word.
Code 11 then indicates that the last symbol encountered (by codes 07 and 10) is an external symbol. The data word contains its own unrelocated address.

12 Global symbol definition.
The last symbol encountered is a global symbol. The corresponding data word contains the unrelocated address of the global symbol.

23 Program name or internal symbol def.
If bit 0 of the data word is 1, then the last symbol encountered is the name of the program unit, otherwise the data word is the address of an internal symbol, which is not relevant to LOAD8.

27 End of program unit.

4.4. Character Representation in the Assembler Output.

The MACRO-15 assembler uses a special character representation in the symbol tables and the output, called RADIX 50 [7]. Three characters, plus a symbol classification bit are grouped together in each 18 bit word. A symbol is defined as a string of one to six characters from the following set:

Character	Octal code
space	00
A	01
↓	
Z	32
%	33
.	34
0	35
↓	
9	46
≠	47

The characters in a symbol are linked together in the following manner:

Word 1 $(C_1*50_8+C_2)*50_8 + C_3$

Word 2 $(C_4*50_8+C_5)*50_8 + C_6$

The C_i:s are the character codes from the table above. If the symbol has less than four characters word 2 is not used and bit 0 of word 1 is 0, otherwise bit 0 of word 1 is 1 and word 2 is used.

5. IMPLEMENTATION OF LOAD4 AND LOAD8.

LOAD4 and LOAD8 are very similar and have most subroutines in common. In the following only LOAD8 is mentioned, but almost everything applies to LOAD4 as well, with the few exceptions explicitly stated.

Most of the routines of LOAD8 are written in FORTRAN. This does not mean, however, that they are in any sense portable. The whole scheme relies on the specific form of the output from the MACRO-15 assembler. Therefore special features of the PDP-15 FORTRAN dialect has been used freely. This applies especially to the DOUBLE INTEGER data representation and to the powerful partword notation of the PDP-15 FORTRAN.

LOAD8 is a two-pass loader in the sense that the binary files from the assembler are scanned twice. The first pass collects information of global symbols, external references, size etc and the second pass performs the relocation using the previously collected information. Due to the two-pass structure no intermediate form of output is necessary neither in core nor on disk. In pass two the output words can be generated directly when an input word containing a machine instruction or address is read from the input files.

5.1. Data Base for LOAD8.

The data base for LOAD8 consists essentially of the following INTEGER and DOUBLE INTEGER vectors:

FILNAM Double integer vector containing the names of the program files. The character representation is the so called 5/7 ASCll used by the PDP-15 system.

GLOB Double integer vector with the global symbols. The character representation is RADIX50.

IADR Integer vector with the final addresses of the global symbols.

EXTREF Double integer vector with the names of all the external references.
Representation RADIX50.

NAME Double integer vector with the names of the program units. This is not the same as a program file, since a program file may contain more than one program unit.
Representation RADIX50.

IST Integer vector with the start address of each program unit.

NEX Integer vector with one entry per program unit. NEX[I] points to the element of EXTREF which is the first external reference for program unit number I.

NOMTCH Double integer vector containing the entries of EXTREF which does not match an entry of GLOB, i.e. the unresolved globals.

5.2. Program Description.

Fig. 5.1 shows the program structure, and this section describes each routine in some detail.

MAIN8 is the main program of LOAD8. Its task is, of course, to administrate the calls to subroutines. Furthermore, it opens and closes files for the first pass, it asks the operator for the addresses of any unresolved globals and it prints the load map and global symbol table.

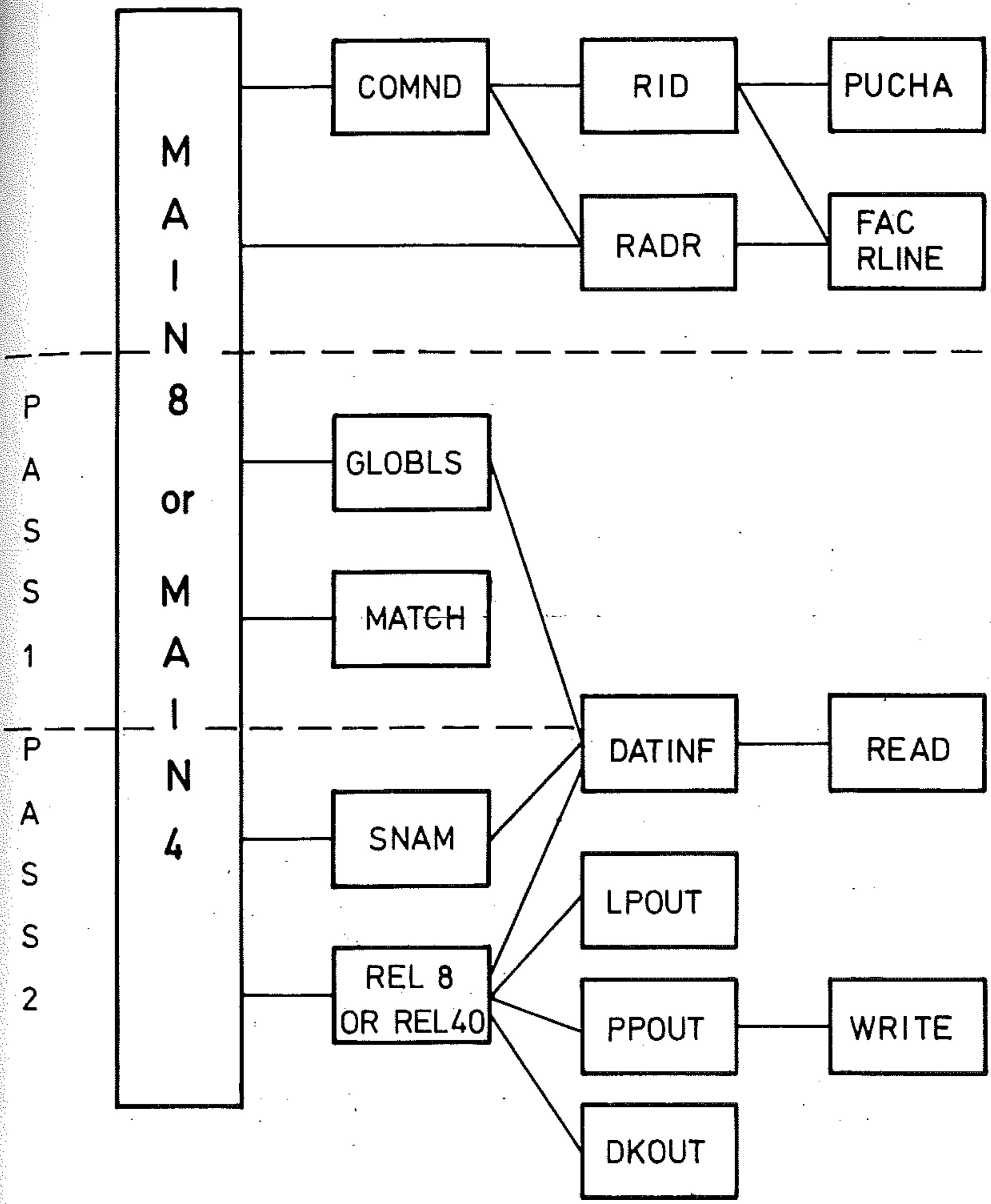

Fig. 5.1 LOAD8 Program structure.

MAIN4 is the main program of LOAD4. The only difference between MAIN4 and MAIN8 is that they call different relocation subroutines, REL4 and REL8 respectively.

COMND reads and decodes the command string using the string-decoding routines RADR, RID, FAC, PUTCHA. As a result the vector FILNAM gets its elements, IST(1) gets a value and some entries of GLOB and IADR may get their values.

DATINF unpacks the information blocks from the binary files produced by the assembler (see sect. 4.3). Each call returns one data word and its corresponding information code.

GLOBLS scans the binary files and extracts information of program unit size, program unit name, global symbols and external references. The vectors GLOB, IADR, EXTREF, NAME, IST and NEX get their values from this routine.

MATCH compares the vectors GLOB and EXTREF. The entries of EXTREF which do not exist in GLOB are the unresolved globals which are collected in the vector NOMTCH.

SNAM scans the input files for a given program unit name. When it encounters an end of file, it closes that file and opens next as given by the entries of the vector FILNAM. Since the files are read twice in the same order, and since the names given to SNAM will be taken from the vector NAME in the order they were encountered at the first reading, SNAM will always find the requested name by reading ahead in the files. When SNAM exits the input file is positioned at the beginning of the requested program unit. The main use of SNAM is in library search, where some program units should be loaded and some be skipped.

REL8 performs the actual relocation and linking. It is in fact the only routine which is dependent of the target computer.
REL8 utilizes the fact that the assembler will output one extra word for each external reference (see sect.

4.3, code 11). The size information given by the assembler naturally includes these extra words. However, when GLOBLS scans the program unit, the number of external references is subtracted from the size, so that when the start address of the next program unit is computed, these extra words will not be included.
REL8 can then check each relocatable address to see if it points to a position within the program unit. If it does, the address is simply relocated by adding the program unit start address. If it points outside the program unit, it is an external reference. The symbolic name of this reference can be found in EXTREF, since its entries appear in the same order as the corresponding extra words. When the name is found, GLOBLS is scanned for it, and the final address is found in the corresponding entry of the vector IADR.

REL4 is the corresponding relocation subroutine for the 4040. The relocation and search for external references is the same as REL8. REL4 has the additional task of checking for JCN and ISZ instructions, because these take only eight bits address. The destination must then be on the same page as the instruction itself.

READ and WRITE are FORTRAN callable subroutines to read or write in any data mode. In this case we need to read in the IOPS Binary mode and write in the Image Alphanumeric mode.

5.3. Program Flow.

When the command strings have been decoded by COMND, the subroutine GLOBLS scans all the program files. Then MATCH is called to check if any unresolved external references exist. If so, and if a library file has been specified, it is scanned by GLOBLS. After each program unit in the library MATCH is called again. If any external reference has been resolved by this unit its name is entered into NAME and its global symbols and external references are entered into GLOB, IADR and EXTREF. When the library is exhausted MATCH checks if any external references are still unresolved. If so the operator is asked to give the addresses. This concludes Pass 1.

In Pass 2 the program files and the library file are scanned again, this time by SNAM, which positions the file to the beginning of each program unit, and by REL8, which performs the actual relocation and linking and also calls the proper output routines.

6. EXTENSIONS.

It is quite simple to extend this programming tool to other target machines. The definition file, i.e. the file that corresponds to ASS8, ASS80 or ASS40 must be written, but this is a reasonable task since most microprocessors have a rather simple instruction set. The subroutine REL8 would also have to be rewritten - and renamed - but it is a straightforward routine with some 45 FORTRAN statements. As long as the word length of the target machine is 8 bits these changes are sufficient.

If the word length of the target machine is 16 bits, the input and output format would have to be changed. This affects MAIN8, RADR, LPOUT, PPOUT and DKOUT.

BASIC INSTRUCTION SET

The basic instruction set of the 4040 and 4004 (CPU) are shown below. The following section will describe each instruction in detail.

[Those instructions preceded by an asterisk (*) are 2 word instructions that occupy 2 successive locations in ROM]

MACHINE INSTRUCTIONS (Logic 1 = Low Voltage = Negative Voltage; Logic 0 = High Voltage = Ground)

MNEMONIC	OPR $D_3 D_2 D_1 D_0$	OPA $D_3 D_2 D_1 D_0$	DESCRIPTION OF OPERATION
NOP	0 0 0 0	0 0 0 0	No operation.
*JCN	0 0 0 1 $A_2 A_2 A_2 A_2$	$C_1 C_2 C_3 C_4$ $A_1 A_1 A_1 A_1$	Jump to ROM address $A_2 A_2 A_2 A_2$, $A_1 A_1 A_1 A_1$ (within the same ROM that contains this JCN instruction) if condition $C_1 C_2 C_3 C_4$ (1) is true, otherwise skip (go to the next instruction in sequence).
*FIM	0 0 1 0 $D_2 D_2 D_2 D_2$	R R R 0 $D_1 D_1 D_1 D_1$	Fetch immediate (direct) from ROM Data D_2, D_1 to index register pair location RRR. (2)
SRC	0 0 1 0	R R R 1	Send register control. Send the address (contents of index register pair RRR) to ROM and RAM at X_2 and X_3 time in the Instruction Cycle.
FIN	0 0 1 1	R R R 0	Fetch indirect from ROM. Send contents of index register pair location 0 out as an address. Data fetched is placed into register pair location RRR.
JIN	0 0 1 1	R R R 1	Jump indirect. Send contents of register pair RRR out as an address at A_1 and A_2 time in the Instruction Cycle.
*JUN	0 1 0 0 $A_2 A_2 A_2 A_2$	$A_3 A_3 A_3 A_3$ $A_1 A_1 A_1 A_1$	Jump unconditional to ROM address A_3, A_2, A_1.
*JMS	0 1 0 1 $A_2 A_2 A_2 A_2$	$A_3 A_3 A_3 A_3$ $A_1 A_1 A_1 A_1$	Jump to subroutine ROM address A_3, A_2, A_1, save old address. (Up 1 level in stack.)
INC	0 1 1 0	R R R R	Increment contents of register RRRR. (3)
*ISZ	0 1 1 1 $A_2 A_2 A_2 A_2$	R R R R $A_1 A_1 A_1 A_1$	Increment contents of register RRRR. Go to ROM address A_2, A_1 (within the same ROM that contains this ISZ instruction) if result ≠0, otherwise skip (go to the next instruction in sequence).
ADD	1 0 0 0	R R R R	Add contents of register RRRR to accumulator with carry.
SUB	1 0 0 1	R R R R	Subtract contents of register RRRR to accumulator with borrow.
LD	1 0 1 0	R R R R	Load contents of register RRRR to accumulator.
XCH	1 0 1 1	R R R R	Exchange contents of index register RRRR and accumulator.
BBL	1 1 0 0	D D D D	Branch back (down 1 level in stack) and load data DDDD to accumulator.
LDM	1 1 0 1	D D D D	Load data DDDD to accumulator.

NEW 4040 INSTRUCTIONS

MNEMONIC	OPR $D_3 D_2 D_1 D_0$	OPA $D_3 D_2 D_1 D_0$	DESCRIPTION OF OPERATION
HLT	0 0 0 0	0 0 0 1	Halt – inhibit program counter and data buffers.
BBS	0 0 0 0	0 0 1 0	Branch Back from Interrupt and restore the previous SRC. The Program Counter and send register control are restored to their pre-interrupt value.
LCR	0 0 0 0	0 0 1 1	The contents of the COMMAND REGISTER are transferred to the ACCUMULATOR.
OR4	0 0 0 0	0 1 0 0	The 4 bit contents of register #4 are logically "OR-ed" with the ACCUM.
OR5	0 0 0 0	0 1 0 1	The 4 bit contents of index register #5 are logically "OR-ed" with the ACCUMULATOR.
AN6	0 0 0 0	0 1 1 0	The 4 bit contents of index register #6 are logically "AND-ed" with the ACCUMULATOR
AN7	0 0 0 0	0 1 1 1	The 4 bit contents of index register #7 are logically "AND-ed" with the ACCUMULATOR.
DB0	0 0 0 0	1 0 0 0	DESIGNATE ROM BANK 0. CM-ROM_0 becomes enabled.
DB1	0 0 0 0	1 0 0 1	DESIGNATE ROM BANK 1. CM-ROM_1 becomes enabled.
SB0	0 0 0 0	1 0 1 0	SELECT INDEX REGISTER BANK 0. The index registers 0 - 7.
SB1	0 0 0 0	1 0 1 1	SELECT INDEX REGISTER BANK 1. The index registers 0* - 7*.
EIN	0 0 0 0	1 1 0 0	ENABLE INTERRUPT.
DIN	0 0 0 0	1 1 0 1	DISABLE INTERRUPT.
RPM	0 0 0 0	1 1 1 0	READ PROGRAM MEMORY.

INPUT/OUTPUT AND RAM INSTRUCTIONS

(The RAM's and ROM's operated on in the I/O and RAM instructions have been previously selected by the last SRC instruction executed.)

MNEMONIC	OPR $D_3 D_2 D_1 D_0$	OPA $D_3 D_2 D_1 D_0$	DESCRIPTION OF OPERATION
WRM	1 1 1 0	0 0 0 0	Write the contents of the accumulator into the previously selected RAM main memory character.
WMP	1 1 1 0	0 0 0 1	Write the contents of the accumulator into the previously selected RAM output port.
WRR	1 1 1 0	0 0 1 0	Write the contents of the accumulator into the previously selected ROM output port. (I/O Lines)
WPM	1 1 1 0	0 0 1 1	Write the contents of the accumulator into the previously selected half byte of read/write program memory (for use with 4008/4009 only)
WRφ (4)	1 1 1 0	0 1 0 0	Write the contents of the accumulator into the previously selected RAM status character 0.
WR1 (4)	1 1 1 0	0 1 0 1	Write the contents of the accumulator into the previously selected RAM status character 1.
WR2 (4)	1 1 1 0	0 1 1 0	Write the contents of the accumulator into the previously selected RAM status character 2.
WR3 (4)	1 1 1 0	0 1 1 1	Write the contents of the accumulator into the previously selected RAM status character 3.
SBM	1 1 1 0	1 0 0 0	Subtract the previously selected RAM main memory character from accumulator with borrow.
RDM	1 1 1 0	1 0 0 1	Read the previously selected RAM main memory character into the accumulator.
RDR	1 1 1 0	1 0 1 0	Read the contents of the previously selected ROM input port into the accumulator. (I/O Lines)
ADM	1 1 1 0	1 0 1 1	Add the previously selected RAM main memory character to accumulator with carry.
RDφ (4)	1 1 1 0	1 1 0 0	Read the previously selected RAM status character 0 into accumulator.
RD1 (4)	1 1 1 0	1 1 0 1	Read the previously selected RAM status character 1 into accumulator.
RD2 (4)	1 1 1 0	1 1 1 0	Read the previously selected RAM status character 2 into accumulator.
RD3 (4)	1 1 1 0	1 1 1 1	Read the previously selected RAM status character 3 into accumulator.

ACCUMULATOR GROUP INSTRUCTIONS

CLB	1 1 1 1	0 0 0 0	Clear both. (Accumulator and carry)
CLC	1 1 1 1	0 0 0 1	Clear carry.
IAC	1 1 1 1	0 0 1 0	Increment accumulator.
CMC	1 1 1 1	0 0 1 1	Complement carry.
CMA	1 1 1 1	0 1 0 0	Complement accumulator.
RAL	1 1 1 1	0 1 0 1	Rotate left. (Accumulator and carry)
RAR	1 1 1 1	0 1 1 0	Rotate right. (Accumulator and carry)
TCC	1 1 1 1	0 1 1 1	Transmit carry to accumulator and clear carry.
DAC	1 1 1 1	1 0 0 0	Decrement accumulator.
TCS	1 1 1 1	1 0 0 1	Transfer carry subtract and clear carry.
STC	1 1 1 1	1 0 1 0	Set carry.
DAA	1 1 1 1	1 0 1 1	Decimal adjust accumulator.
KBP	1 1 1 1	1 1 0 0	Keyboard process. Converts the contents of the accumulator from a one out of four code to a binary code.
DCL	1 1 1 1	1 1 0 1	Designate command line.

NOTES: (1) The condition code is assigned as follows:

$C_1 = 1$ Invert jump condition; $C_2 = 1$ Jump if accumulator is zero; $C_4 = 1$ Jump if test signal is a 0
$C_1 = 0$ Not invert jump condition; $C_3 = 1$ Jump if carry/link is a 1

(2) RRR is the address of 1 of 8 index register pairs in the CPU.

(3) RRRR is the address of 1 of 16 index registers in the CPU.

(4) Each RAM chip has 4 registers; each with twenty 4-bit characters subdivided into 16 main memory characters and 4 status characters. Chip number, RAM register and main memory character are addressed by an SRC instruction. For the selected chip and register, however, status character locations are selected by the instruction code (OPA).

IV. BASIC INSTRUCTION SET

The following section presents the basic instruction set of the 8008. For a detailed description of the execution of each instruction, refer to Appendix I.

Data and Instruction Formats

Data in the 8008 is stored in the form of 8-bit binary integers. All data transfers to the system data bus will be in the same format.

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

DATA WORD

The program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive words in program memory. The instruction formats then depend on the particular operation executed.

		TYPICAL INSTRUCTIONS
One Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	Register to register, memory reference, I/O arithmetic or logical, rotate or return instructions
Two Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	Immediate mode instructions
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OPERAND	
Three Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	JUMP or CALL instructions
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	LOW ADDRESS	
X X D_5 D_4 D_3 D_2 D_1 D_0	HIGH ADDRESS*	

*For the third byte of this instruction, D_6 and D_7 are "don't care" bits.

For the MCS-8 a logic "1" is defined as a high level and a logic "0" is defined as a low level.

Index Register Instructions

The load instructions do not affect the flag flip-flops. The increment and decrement instructions affect all flip-flops except the carry.

MNEMONIC	MINIMUM STATES REQUIRED	INSTRUCTION CODE $D_7 D_6$	$D_5 D_4 D_3$	$D_2 D_1 D_0$	DESCRIPTION OF OPERATION
(1) Lr_1r_2	(5)	1 1	D D D	S S S	Load index register r_1 with the content of index register r_2.
(2) LrM	(8)	1 1	D D D	1 1 1	Load index register r with the content of memory register M.
LMr	(7)	1 1	1 1 1	S S S	Load memory register M with the content of index register r.
(3) LrI	(8)	0 0 B B	D D D B B B	1 1 0 B B B	Load index register r with data B . . . B.
LMI	(9)	0 0 B B	1 1 1 B B B	1 1 0 B B B	Load memory register M with data B . . . B.
INr	(5)	0 0	D D D	0 0 0	Increment the content of index register r ($r \neq A$).
DCr	(5)	0 0	D D D	0 0 1	Decrement the content of index register r ($r \neq A$).

Accumulator Group Instructions

The result of the ALU instructions affect all of the flag flip-flops. The rotate instructions affect only the carry flip-flop.

MNEMONIC	MINIMUM STATES REQUIRED	$D_7 D_6$	$D_5 D_4 D_3$	$D_2 D_1 D_0$	DESCRIPTION OF OPERATION
ADr	(5)	1 0	0 0 0	S S S	Add the content of index register r, memory register M, or data B . . . B to the accumulator. An overflow (carry) sets the carry flip-flop.
ADM	(8)	1 0	0 0 0	1 1 1	
ADI	(8)	0 0 B B	0 0 0 B B B	1 0 0 B B B	
ACr	(5)	1 0	0 0 1	S S S	Add the content of index register r, memory register M, or data B . . . B from the accumulator with carry. An overflow (carry) sets the carry flip-flop.
ACM	(8)	1 0	0 0 1	1 1 1	
ACI	(8)	0 0 B B	0 0 1 B B B	1 0 0 B B B	
SUr	(5)	1 0	0 1 0	S S S	Subtract the content of index register r, memory register M, or data B . . . B from the accumulator. An underflow (borrow) sets the carry flip-flop.
SUM	(8)	1 0	0 1 0	1 1 1	
SUI	(8)	0 0 B B	0 1 0 B B B	1 0 0 B B B	
SBr	(5)	1 0	0 1 1	S S S	Subtract the content of index register r, memory register M, or data B . . . B from the accumulator with borrow. An underflow (borrow) sets the carry flip-flop.
SBM	(8)	1 0	0 1 1	1 1 1	
SBI	(8)	0 0 B B	0 1 1 B B B	1 0 0 B B B	
NDr	(5)	1 0	1 0 0	S S S	Compute the logical AND of the content of index register r, memory register M, or data B . . . B with the accumulator.
NDM	(8)	1 0	1 0 0	1 1 1	
NDI	(8)	0 0 B B	1 0 0 B B B	1 0 0 B B B	
XRr	(5)	1 0	1 0 1	S S S	Compute the EXCLUSIVE OR of the content of index register r, memory register M, or data B . . . B with the accumulator.
XRM	(8)	1 0	1 0 1	1 1 1	
XRI	(8)	0 0 B B	1 0 1 B B B	1 0 0 B B B	
ORr	(5)	1 0	1 1 0	S S S	Compute the INCLUSIVE OR of the content of index register r, memory register m, or data B . . . B with the accumulator.
ORM	(8)	1 0	1 1 0	1 1 1	
ORI	(8)	0 0 B B	1 1 0 B B B	1 0 0 B B B	
CPr	(5)	1 0	1 1 1	S S S	Compare the content of index register r, memory register M, or data B . . . B with the accumulator. The content of the accumulator is unchanged.
CPM	(8)	1 0	1 1 1	1 1 1	
CPI	(8)	0 0 B B	1 1 1 B B B	1 0 0 B B B	
RLC	(5)	0 0	0 0 0	0 1 0	Rotate the content of the accumulator left.
RRC	(5)	0 0	0 0 1	0 1 0	Rotate the content of the accumulator right.
RAL	(5)	0 0	0 1 0	0 1 0	Rotate the content of the accumulator left through the carry.
RAR	(5)	0 0	0 1 1	0 1 0	Rotate the content of the accumulator right through the carry.

Program Counter and Stack Control Instructions

MNEMONIC	MINIMUM STATES REQUIRED	$D_7 D_6$	$D_5 D_4 D_3$	$D_2 D_1 D_0$	DESCRIPTION OF OPERATION
(4) JMP	(11)	0 1 B_2 B_2 X X	X X X B_2 B_2 B_2 B_3 B_3 B_3	1 0 0 B_2 B_2 B_2 B_3 B_3 B_3	Unconditionally jump to memory address $B_3 \ldots B_3B_2 \ldots B_2$.
(5) JFc	(9 or 11)	0 1 B_2 B_2 X X	0 C_4 C_3 B_2 B_2 B_2 B_3 B_3 B_3	0 0 0 B_2 B_2 B_2 B_3 B_3 B_3	Jump to memory address $B_3 \ldots B_3B_2 \ldots B_2$ if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence.
JTc	(9 or 11)	0 1 B_2 B_2 X X	1 C_4 C_3 B_2 B_2 B_2 B_3 B_3 B_3	0 0 0 B_2 B_2 B_2 B_3 B_3 B_3	Jump to memory address $B_3 \ldots B_3B_2 \ldots B_2$ if the condition flip-flop c is true. Otherwise, execute the next instruction in sequence.
CAL	(11)	0 1 B_2 B_2 X X	X X X B_2 B_2 B_2 B_3 B_3 B_3	1 1 0 B_2 B_2 B_2 B_3 B_3 B_3	Unconditionally call the subroutine at memory address $B_3 \ldots B_3B_2 \ldots B_2$. Save the current address (up one level in the stack).
CFc	(9 or 11)	0 1 B_2 B_2 X X	0 C_4 C_3 B_2 B_2 B_2 B_3 B_3 B_3	0 1 0 B_2 B_2 B_2 B_3 B_3 B_3	Call the subroutine at memory address $B_3 \ldots B_3B_2 \ldots B_2$ if the condition flip-flop c is false, and save the current address (up one level in the stack.) Otherwise, execute the next instruction in sequence.
CTc	(9 or 11)	0 1 B_2 B_2 X X	1 C_4 C_3 B_2 B_2 B_2 B_3 B_3 B_3	0 1 0 B_2 B_2 B_2 B_3 B_3 B_3	Call the subroutine at memory address $B_3 \ldots B_3B_2 \ldots B_2$ if the condition flip-flop c is true, and save the current address (up one level in the stack). Otherwise, execute the next instruction in sequence.
RET	(5)	0 0	X X X	1 1 1	Unconditionally return (down one level in the stack).
RFc	(3 or 5)	0 0	0 C_4 C_3	0 1 1	Return (down one level in the stack) if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence.
RTc	(3 or 5)	0 0	1 C_4 C_3	0 1 1	Return (down one level in the stack) if the condition flip-flop c is true. Otherwise, execute the next instruction in sequence.
RST	(5)	0 0	A A A	1 0 1	Call the subroutine at memory address AAA000 (up one level in the stack).

Input/Output Instructions

MNEMONIC	MINIMUM STATES REQUIRED	$D_7 D_6$	$D_5 D_4 D_3$	$D_2 D_1 D_0$	DESCRIPTION OF OPERATION
INP	(8)	0 1	0 0 M	M M 1	Read the content of the selected input port (MMM) into the accumulator.
OUT	(6)	0 1	R R M	M M 1	Write the content of the accumulator into the selected output port (RRMMM, RR ≠ 00).

Machine Instruction

MNEMONIC	MINIMUM STATES REQUIRED	$D_7 D_6$	$D_5 D_4 D_3$	$D_2 D_1 D_0$	DESCRIPTION OF OPERATION
HLT	(4)	0 0	0 0 0	0 0 X	Enter the STOPPED state and remain there until interrupted.
HLT	(4)	1 1	1 1 1	1 1 1	Enter the STOPPED state and remain there until interrupted.

NOTES:

(1) SSS = Source Index Register, DDD = Destination Index Register — These registers, r_i, are designated A(accumulator—000), B(001), C(010), D(011), E(100), H(101), L(110).

(2) Memory registers are addressed by the contents of registers H & L.

(3) Additional bytes of instruction are designated by BBBBBBBB.

(4) X = "Don't Care".

(5) Flag flip-flops are defined by C_4C_3: carry (00-overflow or underflow), zero (01-result is zero), sign (10-MSB of result is "1"), parity (11-parity is even).

SILICON GATE MOS 8080A

INSTRUCTION SET

Summary of Processor Instructions

Mnemonic	Description	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
		Instruction Code[1]								
MOV r1,r2	Move register to register	0	1	D	D	D	S	S	S	5
MOV M,r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r,M	Move memory to register	0	1	D	D	D	1	1	0	7
HLT	Halt	0	1	1	1	0	1	1	0	7
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
INR r	Increment register	0	0	D	D	D	1	0	0	5
DCR r	Decrement register	0	0	D	D	D	1	0	1	5
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
ORA r	Or register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	10
JZ	Jump on zero	1	1	0	0	1	0	1	0	10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	10
JP	Jump on positive	1	1	1	1	0	0	1	0	10
JM	Jump on minus	1	1	1	1	1	0	1	0	10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	10
CALL	Call unconditional	1	1	0	0	1	1	0	1	17
CC	Call on carry	1	1	0	1	1	1	0	0	11/17
CNC	Call on no carry	1	1	0	1	0	1	0	0	11/17
CZ	Call on zero	1	1	0	0	1	1	0	0	11/17
CNZ	Call on no zero	1	1	0	0	0	1	0	0	11/17
CP	Call on positive	1	1	1	1	0	1	0	0	11/17
CM	Call on minus	1	1	1	1	1	1	0	0	11/17
CPE	Call on parity even	1	1	1	0	1	1	0	0	11/17
CPO	Call on parity odd	1	1	1	0	0	1	0	0	11/17
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	5/11
RNC	Return on no carry	1	1	0	1	0	0	0	0	5/11

Mnemonic	Description	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
		Instruction Code[1]								
RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
RNZ	Return on no zero	1	1	0	0	0	0	0	0	5/11
RP	Return on positive	1	1	1	1	0	0	0	0	5/11
RM	Return on minus	1	1	1	1	1	0	0	0	5/11
RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
RST	Restart	1	1	A	A	A	1	1	1	11
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	11
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
POP B	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	18
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	5
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	5
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	5
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	5
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	5
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	5
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	5
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	5
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
EI	Enable Interrupts	1	1	1	1	1	0	1	1	4
DI	Disable interrupt	1	1	1	1	0	0	1	1	4
NOP	No-operation	0	0	0	0	0	0	0	0	4

NOTES:
1. DDD or SSS – 000 B – 001 C – 010 D – 011 E – 100 H – 101 L – 110 Memory – 111 A.
2. Two possible cycle times, (5/11) indicate instruction cycles dependent on condition flags.

```
        .NOLST
HLT=1
BBS=2
LCR=3
OR4=4
OR5=5
AN6=6
AN7=7
DB0=10
DB1=11
SB0=12
SB1=13
EIN=14
DIN=15
RPM=16
WRM=340
WMP=341
WRR=342
WPM=343
WR0=344
WR1=345
WR2=346
WR3=347
SBM=350
RDM=351
RDR=352
ADM=353
RD0=354
RD1=355
RD2=356
RD3=357
CLB=360
CLC=361
IAC=362
CMC=363
CMA=364
RAL=365
RAR=366
TCC=367
DAC=370
TCS=371
STC=372
DAA=373
KBP=374
DCL=375
NOP=0
R0=0
R1=1
R2=2
R3=3
R4=4
R5=5
R6=6
R7=7
R8=10
R9=11
R10=12
R11=13
R12=14
R13=15
R14=16
```

```
R15=17
        .DEFIN  INC,R
        17&R+140
        .ENDM
        .DEFIN  ADD,R
        17&R+200
        .ENDM
        .DEFIN  SUB,R
        17&R+220
        .ENDM
        .DEFIN  LD,R
        17&R+240
        .ENDM
        .DEFIN  XCH,R
        17&R+260
        .ENDM
        .DEFIN  SRC,R
        16&R+41
        .ENDM
        .DEFIN  FIN,R
        16&R+60
        .ENDM
        .DEFIN  JIN,R
        16&R+61
        .ENDM
        .DEFIN  ISZ,R,A
        R&17+160
        A
        .ENDM
        .DEFIN  JMS,A
        120
        A
        .ENDM
        .DEFIN  JUN,A
        100
        A
        .ENDM
        .DEFIN  FIM,R,DATA
        R&16+40
        DATA
        .ENDM
        .DEFIN  BBL,DATA
        DATA&17+300
        .ENDM
        .DEFIN  LDM,DATA
        DATA&17+320
        .ENDM
        .DEFIN  JTT,A
        21
        A
        .ENDM
        .DEFIN  JTC,A
        22
        A
        .ENDM
        .DEFIN  JTTC,A
        23
        A
        .ENDM
        .DEFIN  JTZ,A
        24
```

```
A
.ENDM
.DEFIN  JTTZ,A
25
A
.ENDM
.DEFIN  JTCZ,A
26
A
.ENDM
.DEFIN  JTTCZ,A
27
A
.ENDM
.DEFIN  JFT,A
31
A
.ENDM
.DEFIN  JFC,A
32
A
.ENDM
.DEFIN  JFTC,A
33
A
.ENDM
.DEFIN  JFZ,A
34
A
.ENDM
.DEFIN  JFTZ,A
35
A
.ENDM
.DEFIN  JFCZ,A
36
A
.ENDM
.DEFIN  JFTCZ,A
37
A
.ENDM
.DEFIN  JMP,A
30
A
.ENDM
.DEFIN  CHANGE,RX,RY,RZ,RW
LD      RX
XCH     RZ
LD      RY
XCH     RW
JMS     MULT4
.ENDM
.LST
```

```
        .NOLST
/SYMBOL AND MACRO DEFINITIONS FOR INTEL 8008 INSTRUCTIONS
LAA=300
LAB=301
LAC=302
LAD=303
LAE=304
LAH=305
LAL=306
LBA=310
LBB=311
LBC=312
LBD=313
LBE=314
LBH=315
LBL=316
LCA=320
LCB=321
LCC=322
LCD=323
LCE=324
LCH=325
LCL=326
LDA=330
LDB=331
LDC=332
LDD=333
LDE=334
LDH=335
LDL=336
LEA=340
LEB=341
LEC=342
LED=343
LEE=344
LEH=345
LEL=346
LHA=350
LHB=351
LHC=352
LHD=353
LHE=354
LHH=355
LHL=356
LLA=360
LLB=361
LLC=362
LLD=363
LLE=364
LLH=365
LLL=366
LAM=307
LBM=317
LCM=327
LDM=337
LEM=347
LHM=357
LLM=367
LMA=370
LMB=371
LMC=372
```

```
LMD=373
LME=374
LMH=375
LML=376
INB=010
INC=020
IND=030
INE=040
INH=050
INL=060
DCB=011
DCC=021
DCD=031
DCE=041
DCH=051
DCL=061
ADA=200
ADB=201
ADC=202
ADD=203
ADE=204
ADH=205
ADL=206
ADM=207
ACA=210
ACB=211
ACC=212
ACD=213
ACE=214
ACH=215
ACL=216
ACM=217
SUA=220
SUB=221
SUC=222
SUD=223
SUE=224
SUH=225
SUL=226
SUM=227
SBA=230
SBB=231
SBC=232
SBD=233
SBE=234
SBE=234
SBH=235
SBL=236
SBM=237
NDA=240
NDB=241
NDC=242
NDD=243
NDE=244
NDH=245
NDL=246
NDM=247
XRA=250
XRB=251
XRC=252
XRD=253
```

```
XRE=254
XRH=255
XRL=256
XRM=257
ORA=260
ORB=261
ORC=262
ORD=263
ORE=264
ORH=265
ORL=266
ORM=267
CPA=270
CPB=271
CPC=272
CPD=273
CPE=274
CPH=275
CPL=276
CPM=277
RLC=002
RRC=012
RAL=022
RAR=032
RET=007
RFC=003
RFZ=013
RFS=023
RFP=033
RTC=043
RTZ=053
RTS=063
RTP=073
RES=005
HLT=000
        .DEFIN  LAI,DAT
         006
         DAT
        .ENDM
        .DEFIN  LBI,DAT
         016
         DAT
        .ENDM
        .DEFIN  LCI,DAT
         026
         DAT
        .ENDM
        .DEFIN  LDI,DAT
         036
         DAT
        .ENDM
        .DEFIN  LEI,DAT
         046
         DAT
        .ENDM
        .DEFIN  LHI,DAT
         056
         DAT
        .ENDM
        .DEFIN  LLI,DAT
         066
```

```
 DAT
.ENDM
.DEFIN  LMI,DAT
 076
 DAT
.ENDM
.DEFIN  ADI,DAT
 004
 DAT
.ENDM
.DEFIN  ACI,DAT
 014
 DAT
.ENDM
.DEFIN  SUI,DAT
 024
 DAT
.ENDM
.DEFIN  SBI,DAT
 034
 DAT
.ENDM
.DEFIN  NDI,DAT
 044
 DAT
.ENDM
.DEFIN  XRI,DAT
 054
 DAT
.ENDM
.DEFIN  ORI,DAT
 064
 DAT
.ENDM
.DEFIN  CPI,DAT
 074
 DAT
.ENDM
.DEFIN  JMP,ADR
 104
 ADR
 .
.ENDM
.DEFIN  JFC,ADR
 100
 ADR
 .
.ENDM
.DEFIN  JFZ,ADR
 110
 ADR
 .
.ENDM
.DEFIN  JFS,ADR
 120
 ADR
 .
.ENDM
.DEFIN  JFP,ADR
 130
 ADR
```

```
.
.ENDM
.DEFIN  JTC,ADR
 140
 ADR
.
.ENDM
.DEFIN  JTZ,ADR
 150
 ADR
.
.ENDM
.DEFIN  JTS,ADR
 160
 ADR
.
.ENDM
.DEFIN  JTP,ADR
 170
 ADR
.
.ENDM
.DEFIN  CAL,ADR
 106
 ADR
.
.ENDM
.DEFIN  CFC,ADR
 102
 ADR
.
.ENDM
.DEFIN  CFZ,ADR
 112
 ADR
.
.ENDM
.DEFIN  CFS,ADR
 122
 ADR
.
.ENDM
.DEFIN  CFP,ADR
 132
 ADR
.
.ENDM
.DEFIN  CTC,ADR
 142
 ADR
.
.ENDM
.DEFIN  CTZ,ADR
 152
 ADR
.
.ENDM
.DEFIN  CTS,ADR
 162
 ADR
.
```

```
.ENDM
.DEFIN  CTP,ADR
 172
 ADR
 .
.ENDM
.DEFIN  INP,PORT
 101,PORT*2
.ENDM
.DEFIN  OUT,PORT
 101,PORT*2
.ENDM
.DEFIN  PHL,ADR
 066
 ADR
 056
 .
.ENDM
.LST
.EOT
```

```
        .NOLST
/       MACRO FILE FOR  INTEL 8080
RLC=007
RRC=017
RAL=027
RAR=037
RET=311
RC=330
RNC=320
RZ=310
RNZ=300
RP=360
RM=370
RPE=350
RPO=340
XCHG=353
XTHL=343
SPHL=371
PCHL=351
CMA=057
STC=067
CMC=077
DAA=047
EI=373
DI=363
NOP=000
HLT=166
A=7
B=0
C=1
D=2
E=3
H=4
L=5
M=6
SP=6
PSW=6
        .DEFIN MOV,R1,R2
        R1*10+R2+100
        .ENDM
        .DEFIN MVI,R,DAT
        R&7*10+6
        DAT
        .ENDM
        .DEFIN INR,R
        R&7*10+4
        .ENDM
        .DEFIN DCR,R
        R&7*10+5
        .ENDM
        .DEFIN ADD,R
        R&7+200
        .ENDM
        .DEFIN ADC,R
        R&7+210
        .ENDM
        .DEFIN SUB,R
        R&7+220
        .ENDM
        .DEFIN SBB,R
        R&7+230
```

```
.ENDM
.DEFIN ANA,R
R&7+240
.ENDM
.DEFIN XRA,R
R&7+250
.ENDM
.DEFIN ORA,R
R&7+260
.ENDM
.DEFIN CMP,R
R&7+270
.ENDM
.DEFIN ADI,DAT
306
DAT
.ENDM
.DEFIN ACI,DAT
316
DAT
.ENDM
.DEFIN SUI,DAT
326
DAT
.ENDM
.DEFIN SBI,DAT
336
DAT
.ENDM
.DEFIN ANI,DAT
346
DAT
.ENDM
.DEFIN XRI,DAT
356
DAT
.ENDM
.DEFIN ORI,DAT
366
DAT
.ENDM
.DEFIN CPI,DAT
376
DAT
.ENDM
.DEFIN JMP,ADR
303
ADR
.
.ENDM
.DEFIN JC,ADR
332
ADR
.
.ENDM
.DEFIN JNC,ADR
322
ADR
.
.ENDM
.DEFIN JZ,ADR
```

```
312
ADR
.
.ENDM
.DEFIN JNZ,ADR
302
ADR
.
.ENDM
.DEFIN JP,ADR
362
ADR
.
.ENDM
.DEFIN JM,ADR
372
ADR
.
.ENDM
.DEFIN JPE,ADR
352
ADR
.
.ENDM
.DEFIN JPO,ADR
342
ADR
.
.ENDM
.DEFIN CALL,ADR
315
ADR
.
.ENDM
.DEFIN CAL,ADR
315
ADR
.
.ENDM
.DEFIN CC,ADR
334
ADR
.
.ENDM
.DEFIN CNC,ADR
324
ADR
.
.ENDM
.DEFIN CZ,ADR
314
ADR
.
.ENDM
.DEFIN CNZ,ADR
304
ADR
.
.ENDM
.DEFIN CP,ADR
364
```

```
ADR
.
.ENDM
.DEFIN CM,ADR
374
ADR
.
.ENDM
.DEFIN CPE,ADR
354
ADR
.
.ENDM
.DEFIN CPO,ADR
344
ADR
.
.ENDM
.DEFIN RST,ADR
ADR&70+307
.ENDM
.DEFIN IN,ADR
333
ADR
.ENDM
.DEFIN OUT,ADR
323
ADR
.ENDM
.DEFIN LXI,R,DAT
R&6*10+1
DAT&377
DAT/400
.ENDM
.DEFIN PUSH,R
R&6*10+305
.ENDM
.DEFIN POP,R
R&6*10+301
.ENDM
.DEFIN STA,ADR
062
ADR&377
ADR/400
.ENDM
.DEFIN LDA,ADR
072
ADR&377
ADR/400
.ENDM
.DEFIN DAD,R
R&6*10+11
.ENDM
.DEFIN STAX,R
R&2*10+2
.ENDM
.DEFIN LDAX,R
R&2*10+12
.ENDM
.DEFIN INX,R
R&6*10+03
```

```
.ENDM
.DEFIN DCX,R
R&6*10+13
.ENDM
.DEFIN SHLD,ADR
042
ADR&377
ADR/400
.ENDM
.DEFIN LHLD,ADR
052
ADR&377
ADR/400
.ENDM
.LST
.EOT
```

```
C       PROGRAM MAIN8
C
C       MAIN PROGRAM FOR LOAD8
C       AUTHOR LEIF ANDERSSON 1974-11-13
C       REVISED LEIF ANDERSSON 1975-07-08
C
C       SUBROUTINES REQUIRED
C               REL8
C               SNAM
C               COMND
C               MATCH
C               GLOBLS
C               DATINF
C               RADASC
C               ASCRAD
C               LPOUT
C               PPOUT
C               DKOUT
C               RADR
C               RID
C               PUTCHA
C               RLINE
C               FAC
C               INIT
C               READ
C               WRITE
C
        DOUBLE INTEGER FILNAM,GLOB,EXTREF,NAME,NOMTCH,FN,DNAM,
        1                BUFF,LIBR,A,B,ALTM
        DIMENSION FILNAM(25),GLOB(100),IADR(100),EXTREF(100),
        1            NAME(100),IST(100),NEX(100),NOMTCH(100),FN(2),
        2            DNAM(2),BUFF(16)
        COMMON FILNAM,GLOB,IADR,EXTREF,NOMTCH,NAME,IST,NEX
        DATA FN(2)/' BIN'/,
        1    DNAM(2)/' ABS'/,
        2    LIBR/'NONE'/,
        3    ALTM/□D764000000000/
C
C       GET COMMANDS
C
 10     WRITE(9,100)
 100    FORMAT(1X'LOAD8 V3B')
        IGLOB=0
        IEX=0
        CALL COMND(LIBR,GLOB,IADR,IGLOB,IST1,FILNAM,NFIL,DNAM(1),IND)
        IG1=IGLOB+1
        IST(1)=IST1
        IU=1
        NEX(1)=1
C
C       GET GLOBAL SYMBOLS AND EXTERNAL REFERENCES
C
        DO 20 I=1,NFIL
        FN(1)=FILNAM(I)
        CALL SEEK(2,FN)
 15     CALL GLOBLS(IST1,NAME(IU),GLOB,IADR,IGLOB,EXTREF,IEX,IEOF)
        IF(IEOF)99,20,16
 16     IU=IU+1
        NEX(IU)=IEX+1
        IST(IU)=IST1
        GO TO 15
```

```
 20     CALL CLOSE(2)
        IG2=IGLOB
C
C       GET THE GLOBALS FOR THE LIBRARY PROGRAMS IF ANY ARE NEEDED
C
        CALL MATCH(EXTREF,IEX,GLOB,IGLOB,NOMTCH,IN,M)
        IF(IN .EQ. 0)GO TO 50
        IF(LIBR .EQ. 'NONE')GO TO 40
        NFIL=NFIL+1
        FILNAM(NFIL)=LIBR
        FN(1)=LIBR
        CALL SEEK(2,FN)
C
 25     IG=IGLOB
        IE=IEX
        IST1=IST(IU)
 26     CALL GLOBLS(IST1,NAME(IU),GLOB,IADR,IG,EXTREF,IE,IEOF)
        IF(IEOF)99,29,27
 27     IN1=IN
        CALL MATCH(NOMTCH,IN1,GLOB,IG,NOMTCH,IN,M)
        IF(M .EQ. 0)GO TO 25
        CALL MATCH(EXTREF,IE,GLOB,IG,NOMTCH,IN,M)
        IU=IU+1
        IEX=IE
        IGLOB=IG
        NEX(IU)=IEX+1
        IST(IU)=IST1
        IF(IN .NE. 0)GO TO 26
 29     CALL CLOSE(2)
        IG2=IGLOB
        IF(IN .EQ. 0)GO TO 50
C
C       GET THE UNRESOLVED GLOBALS IF ANY
C
 40     WRITE(9,110)
 110    FORMAT(1X'UNRESOLVED GLOBALS')
        DO 45 II=1,IN
        CALL RADASC(NOMTCH(II),A,B)
        WRITE(9,120)A,B,ALTM
 120    FORMAT(1X,A5,A5,A1)
 43     CALL RLINE(9,8,BUFF,0)
        I=1
        CALL RADR(I,IAD,IA)
        IF(IA .EQ. 1)GO TO 44
        WRITE(9,130)
 130    FORMAT(1X'SYNTAX ERROR. RETYPE ADDRESS.')
        GO TO 43
 44     IGLOB=IGLOB+1
        GLOB(IGLOB)=NOMTCH(II)
 45     IADR(IGLOB)=IAD
C
C       WRITE LOAD MAP AND GLOBAL SYMBOL TABLE
C
 50     WRITE(6,200)
 200    FORMAT(1X'FILE NAMES')
        NFIL=NFIL-1
        DO 55 I=1,NFIL
 55     WRITE(6,210)FILNAM(I)
 210    FORMAT(5XA5)
C
        WRITE(6,220)LIBR
```

```
 220    FORMAT('0'/1X'LIBRARY:  ',A5)
C
        IL1=MOD(IST(1),256)
        IH1=IST(1)/256
        IST1=IST(IU)-1
        IL2=MOD(IST1,256)
        IH2=IST1/256
        WRITE(6,230)IH1,IL1,IH2,IL2
 230    FORMAT('0'/1X'MEMORY REQUIRED:  ',O3,':',O3,' - ',
        1      O3,':',O3)
C
        WRITE(6,240)
 240    FORMAT('0'/1X'LOAD MAP')
        IU=IU-1
        DO 60 I=1,IU
        CALL RADASC(NAME(I),A,B)
        IL=MOD(IST(I),256)
        IH=IST(I)/256
 60     WRITE(6,250)A,B,IH,IL
 250    FORMAT(5XA5,A5,O3,':',O3)
C
        IF(IG1 .GT. IG2)GO TO 70
        WRITE(6,260)
 260    FORMAT('0'/1X'GLOBAL SYMBOL TABLE')
        DO 65 I=IG1,IG2
        CALL RADASC(GLOB(I),A,B)
        IL=MOD(IADR(I),256)
        IH=IADR(I)/256
 65     WRITE(6,250)A,B,IH,IL
C
 70     WRITE(6,270)
 270    FORMAT('1')
C
C       START RELOCATING
C
        IF(IND .NE. 2)GO TO 73
        CALL ENTER(1,DNAM)
        WRITE(1,230)IH1,IL1,IH2,IL2
 73     IF(IND .NE. 1)GO TO 75
        CALL INIT(7,1)
        IFP=1
        CALL PPOUT(□377,IFP)
C
 75     IV=1
        IFIL=0
        ICUR=IST(1)
        ICHECK=0
        DO 80 I=1,IU
        II=I+1
        CALL SNAM(FILNAM,IFIL,NAME(I),ITRIP,IERR)
        IF(IERR .LT. 0)GO TO 99
        CALL REL8(ICUR,IST(II),GLOB,IADR,EXTREF,NEX(I),
        1          IND,IV,ICHECK,ITRIP,IFP)
        IF(IV .EQ. -1)GO TO 99
        IF(IV .EQ. -2)GO TO 98
 80     CONTINUE
C
        CALL CLOSE(2)
        IFL=-1
        IFD=-1
        CALL LPOUT(I,ICUR,IFL)
```

```
      IF(IND .NE. 2)GO TO 85
      CALL DKOUT(I,IFD)
      CALL CLOSE(1)
 85   IF(IND .NE. 1)GO TO 90
      CALL PPOUT(ICHECK,IFP)
      IFP=-1
      CALL PPOUT(I,IFP)
      CALL CLOSE(7)
 90   WRITE(6,270)
      GO TO 10
C
      READ(2)I
      WRITE(7)I
 99   WRITE(9,300)
 300  FORMAT(1X'READ ERROR')
 98   WRITE(9,310)
 310  FORMAT(1X'CODE ERROR')
      GO TO 10
      END
```

```
      SUBROUTINE REL8(ICUR,ISN,GLOB,IADR,EXTREF,NEX,
     1                 IND,IV,ICHECK,ITRIP,IFP)
C
C     RELOCATION SUBROUTINE FOR LOAD8
C     AUTHOR LEIF ANDDERSSON 1974-10-19
C     REVISED LEIF ANDERSSON 1974-03-27
C
C     THE INPUT AND OUTPUT FILE MUST BE OPENED BEFORE THE CALL
C     AND INIT MUST BE DONE ON THE PP
C
C     ICUR -  CURRENT RELOCATION ADDRESS
C     ISN -   START ADDRESS OF NEXT PROGRAM UNIT
C     GLOB -  DOUBLE INTEGER VECTOR WITH GLOBAL NAMES
C     IADR -  VECTOR WITH THE ADDRESSES OF THE GLOBALS
C     EXTREF- DOUBLE INTEGER VECTOR WITH THE NAMES OF EXTERNAL REFS
C     NEX -   INDEX OF THE FIRST EXTERNAL REFERENCE FOR THE CURRENT
C             PROGRAM UNIT
C     IND -   OUTPUT INDICATOR          0: LP ONLY
C                                       1: PP+LP
C                                       2: DK+LP
C     IV -    SET IV=1 ON THE FIRST CALL. IV IS RETURNED 0
C             NORMALLY, -1 IF READ ERROR AND -2 IF CODE ERROR
C     ICHECK- CHECKSUM
C     ITRIP-  PARAMETER FOR DATINF. DO NOT CHANGE IT.
C     IFP -   THIS IS THE PARAMETER IFP OF PPOUT.
C
C     SUBROUTINES REQUIRED
C             DATINF
C             (READ)
C             LPOUT
C             DKOUT
C             PPOUT
C             (WRITE)
C
      DOUBLE INTEGER GLOB,EXTREF,NAME
      DIMENSION GLOB(1),EXTREF(1),IADR(1)
C
      IST=ICUR
      IF(IV .NE. 1)GO TO 10
      IV=0
      IFL=1
      IFD=1
 10   CALL DATINF(NINF,NDAT,ITRIP,IEOF)
      IF(IEOF)99,99,15
 15   IF(NINF .EQ. 4)GO TO 20
      IF(NINF .EQ. 5)GO TO 30
      GO TO 10
C
C     ABSOLUTE INSTRUCTION OR CONSTANT
C
 20   IDAT=NDAT[10:17]
      GO TO 55
C
C     RLOCATABLE ADDRESS
C
 30   IA=NDAT+IST
      IF(IA .LT. ISN)GO TO 40
C
C     EXTERNAL REFERENCE
C
      IA=IA-ISN+NEX
```

```
      NAME=EXTREF(IA)
      I=0
35    I=I+1
      IF(NAME .NE. GLOB(I))GO TO 35
      IA=IADR(I)
C
C     OUTPUT SECTION
C
C     FIRST WORD OF RELOCATABLE ADDRESS
C
40    IDAT=MOD(IA,256)
      IF(IND .EQ. 1)CALL PPOUT(IDAT,IFP)
      IF(IND .EQ. 2)CALL DKOUT(IDAT,IFD)
      CALL LPOUT(IDAT,ICUR,IFL)
      ICHECK=ICHECK +IDAT
      ICUR=ICUR+1
      IF(ICUR .EQ. ISN)GO TO 98
C
C     SECOND WORD OF RELOCATABLE ADDRESS AND ABSOLUTE
C     INSTRUCTION OR CONSTANT
C
      CALL DATINF(NINF,NDAT,ITRIP,IEOF)
      IF(IEOF .LE. 0)GO TO 98
45    IDAT=NDAT+IST
      IF(NINF .NE. 5 .OR. IDAT .NE. ICUR)GO TO 98
50    IDAT=IA/256
55    IF(IND .EQ. 1)CALL PPOUT(IDAT,IFP)
      IF(IND .EQ. 2)CALL DKOUT(IDAT,IFD)
      CALL LPOUT(IDAT,ICUR,IFL)
      ICHECK=ICHECK+IDAT
      ICUR=ICUR+1
      IF(ICUR .EQ. ISN)RETURN
      GO TO 10
C
C     CODE ERROR
C
98    IV=-2
      RETURN
C
C     READ ERROR
C
99    IV=-1
      RETURN
      END
```

```
      SUBROUTINE SNAM(FILNAM,IFIL,NAME,ITRIP,IERR)
C
C     SUBROUTINE FOR LOAD8
C     AUTHOR LEIF ANDERSSON 1974-10-20
C     REVISED LEIF ANDERSSON 1975-03-27
C
C     THE ROUTINE SEARCHES THE FILES GIVEN BY THE VECTOR FILNAM
C     FOR THE PROGRAM UNIT NAME GIVEN BY NAME
C
C     FILNAM- DOUBLE INTEGER VECTOR WITH FILE NAMES
C     IFIL -  INDEX FOR FILNAM. SET IFIL=0 ON THE FIRST CALL.
C             DO NOT CHANGE IT OTHERWISE
C     NAME -  PROGRAM UNIT NAME
C     ITRIP - PASSED FROM DATINF.
C     IERR -  RETURNED 0 NORMALLY, -1 IF READ ERROR
C
C     SUBROUTINES REQUIRED
C             DATINF
C             READ
C
      DOUBLE INTEGER FILNAM,NAME,FNAM,SYM
      DIMENSION FILNAM(1),FNAM(2)
      DATA FNAM(2)/' BIN'/,LUN/2/
      IERR=0
      IF(IFIL)10,10,20
 10   IFIL=IFIL+1
      FNAM(1)=FILNAM(IFIL)
      CALL SEEK(LUN,FNAM)
      ITRIP=-1
 20   CALL DATINF(NINF,NDAT,ITRIP,IEOF)
      IF(IEOF .LT. 0)GO TO 99
      IF(IEOF .EQ. 0)GO TO 50
      IF(NINF .EQ. 7)GO TO 30
      IF(NINF .EQ. 8)GO TO 35
      IF(NINF .EQ. 19)GO TO 40
      GO TO 20
C
C     SYMBOL - FIRST PART
C
 30   SYM[1:17]=NDAT[1:17]
      SYM[18:35]=0
      GO TO 20
C
C     SYMBOL - SECOND HALF
C
 35   SYM[19:35]=NDAT[1:17]
      GO TO 20
C
C     PROGRAM UNIT NAME?
C
 40   IF(NDAT .LT. 0 .AND. SYM .EQ. NAME)RETURN
      GO TO 20
C
C     END OF FILE
C
 50   CALL CLOSE(LUN)
      GO TO 10
C
C     READ ERROR
C
 99   IERR=-1
```

```
RETURN
END
```

```
      SUBROUTINE COMND(LIBR,GLOB,IADR,IGLOB,IST,FILNAM,NFIL,
     1                 DNAM,IND)
C     SUBROUTINE FOR LOAD8
C     READS AND DECODES COMMAND LINES
C     AUTHOR LEIF ANDERSSON1975-06-27
C     COMMAND SYNTAX SEE REPORT ON LOAD8
C
C     LIBR -  NAME OF LIBRARY FILE
C     GLOB -  DOUBLE INTFGER VECTOR WITH THE GLOBAL NAMES
C     IADR -  VECTOR WITHE THE GLOBAL ADDRESSES
C     IGLOB - LENGTH OF GLOB AND IADR
C     IST -   START ADDRESS
C     FILNAM- DOUBLE INTFGER VECTOR WITH THE FILE NAMES
C     NFIL -  LENGTH OF FILNAM
C     DFIL -  NAME OF RELOCATED FILE ON DISK
C     IND -   OUTPUT INDICATOR
C             1: PP OUTPUT
C             2: DISK OUTPUT
C
C     SUBROUTINES REQUIRED
C             RID
C             RADR
C             FAC
C
      DOUBLE INTEGER LIBR,GLOB,FILNAM,DNAM,BUFF,A,B
      DIMENSION GLOB(1),IADR(1),FILNAM(1),BUFF(16)
C
 10   CALL RLINE(9,8,BUFF,1)
      I=1
      CALL RID(I,A,IT)
      GO TO(15,99,99,99,10,99,99),IT
C
C     CHECK COMMAND
C
 15   IF(A .EQ. 'LIBR')GO TO 20
      IF(A .EQ. 'GLOBL')GO TO 30
      IF(A .EQ. 'LOAD')GO TO 40
      IF(A .EQ. 'STOP')STOP
      GO TO 99
C
C     LIBR COMMAND
C
 20   CALL RID(I,A,IT)
      IF(IT .NE. 1)GO TO 99
      LIBR=A
      GO TO 10
C
C     GLOBL COMMAND
C
 30   IG=IGLOB
 31   CALL RID(I,A,IT)
      B='      '
      GO TO(33,32,34,99,35,36,99),IT
 32   CALL RID2(I,B,IT)
      IF(IT .NE. 1)GO TO 99
 33   IG=IG+1
      CALL RADR(I,IADR(IG),IT)
      IF(IT .NE. 1)GO TO 99
      CALL ASCRAD(A,B,GLOB(IG))
      GO TO 31
 34   IF(A .NE. ',')GO TO 99
```

```
      GO TO 31
 35   CALL RLINE(9,8,BUFF,0)
      I=1
      GO TO 31
 36   IGLOR=IG
      WRITE(9,200)
      GO TO 10
C
C     LOAD COMMAND
C
 40   NFIL=0
      IND=1
 41   CALL RID(I,A,IT)
      GO TO(42,99,99,43,99,99,99),IT
 42   DFIL=A
      IND=2
 43   CALL RADR(I,IST,IT)
      IF(IT .NE. 1)GO TO 99
 44   CALL RID(I,A,IT)
      GO TO(45,99,46,99,47,48,99),IT
 45   NFIL=NFIL+1
      FILNAM(NFIL)=A
      GO TO 44
 46   IF(A .NE. ',')GO TO 99
      GO TO 44
 47   CALL RLINE(9,8,BUFF,0)
      I=1
      GO TO 44
 48   IF(NFIL .EQ. 0)GO TO 99
      WRITE(9,200)
      RETURN
C
C     ERROR MESSAGE
C
 99   WRITE(9,100)
      GO TO 10
 100  FORMAT(1X'SYNTAX ERROR')
 200  FORMAT(1X)
      END
```

```
      SUBROUTINE MATCH(VECT1,NV1,VECT2,NV2,NOMTCH,IN,M)
C     SUBROUTINE FOR LOAD8
C     THE ROUTINE SEARCHES TWO DOUBLE INTEGER VECTORS FOR EQUAL
C     ELEMENTS.
C     AUTHOR LEIF ANDERSSON 1974-07-04
C
C     VECT1 - FIRST VECTOR
C     NV1 -   LENGTH OF VECT1
C     VECT2 - SECOND VECTOR
C     NV2 -   LENGTH OF VECT2
C     NOMTCH- DOUBLE INTEGER VECTOR RETURNED CONTAINING THE ENTRIES
C             OF VECT1 WHICH DO NOT MATCH ANY ELEMENT OF VECT2
C     IN -    LENGTH OF NOMTCH
C     M -     NUMBER OF ELEMENTS OF VECT1 WHICH MATCH ELEMENTS OF VECT
C
C     SUBROUTINES REQUIRED
C             NONE
C
      DOUBLE INTEGER VECT1(1),VECT2(1),NOMTCH(1)
C
      M=0
      IN=0
      IF(NV1 .EQ. 0)RETURN
      IF(NV2 .NE. 0)GO TO 20
      IN=NV1
      DO 10 I=1,IN
   10 NOMTCH(I)=VECT1(I)
      RETURN
C
   20 DO 40 I=1,NV1
      DO 30 J=1,NV2
      IF(VECT1(I) .EQ. VECT2(J))GO TO 35
   30 CONTINUE
      IN=IN+1
      NOMTCH(IN)=VECT1(I)
      GO TO 40
   35 M=M+1
   40 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE GLOBLS(IST,NAME,GLOB,IADR,IGLOB,EXTREF,IEX,IEOF)
C
C     SUBROUTINE FOR LOAD8
C     THE ROUTINE EXTRACTS GLOBAL DEFINITIONS AND EXTERNAL REFERENCES
C     FROM AN ASSEMBLY FILE AND GIVES THE GLOBALS THEIR FINAL
C     ADDRESSES. THE FILE MUST BE OPENED BEFORE THE CALL.
C     AUTHOR LEIF ANDERSSON 1974-07-04
C     REVISED LEIF ANDERSSON 1975-01-30
C
C     IST-    START ADDRESS FOR THE PROGRAM UNIT. WILL BE RETURNED
C             CONTAINING THE START ADDRESS OF THE NEXT UNIT.
C     NAME-    DOUBLE INTEGER RETURNED CONTAINING THE NAME OF THE PRO(
C             RAM UNIT IN RADIX50
C     GLOB-   DOUBLE INTEGER VECTOR CONTAINING THE NAMES OF THE GLOBAl
C     IADR-   VECTOR CONTAINING THE FINAL ADDRESSES OF THE GLOBALS.
C     IGLOB-  INDEX FOR THE VECTORS GLOB AND IADR
C     EXTREF- DOUBLE INTEGER VECTOR CONTAINING THE NAMES OF THE
C             EXTERNAL REFERENCES
C     IEX-    INDEX FOR EXTREF
C     IEOF-   RETURNED POSITIVE NORMALLY, 0 IF END OF FILE AND
C             NEGATIVE IF READ ERROR
C
C     SUBROUTINES REQUIRED
C             DATINF
C             READ
C
      DOUBLE INTEGER NAME,GLOB,EXTREF,SYMBOL
      DIMENSION GLOB(1),EXTREF(1),IADR(1),ISYM(2)
      EQUIVALENCE(SYMBOL,ISYM(1))
C
      ITRIP=-1
      NEX=0
 10   CALL DATINF(INF,IDAT,ITRIP,IEOF)
      IF(IEOF .LE. 0)RETURN
      IF(INF .EQ. 1)GO TO 20
      IF(INF .EQ. 7)GO TO 30
      IF(INF .EQ. 8)GO TO 40
      IF(INF .EQ. 9)GO TO 50
      IF(INF .EQ. 10)GO TO 60
      IF(INF .EQ. 19)GO TO 70
      IF(INF .EQ. 23)GO TO 80
      GO TO 10
C
C     PROGRAM SIZE
C
 20   ISIZE=IDAT
      GO TO 10
C
C     SYMBOL - FIRST HALF
C
 30   ISYM(1)=IDAT[1:17]
      ISYM(2)=0
      GO TO 10
C
C     SYMBOL - SECOND HALF
C
 40   ISYM(2)=IDAT
      GO TO 10
C
C     EXTERNAL REFERENCE
C
```

```
 50     IEX=IEX+1
        NEX=NEX+1
        EXTREF(IEX)=SYMBOL
        GO TO 10
C
C       GLOBL DEFINITION
C
 60     IGLOB=IGLOB+1
        GLOB(IGLOB)=SYMBOL
        IADR(IGLOB)=IST+IDAT
        GO TO 10
C
C       PROGRAM UNIT NAME
C
 70     IF(IDAT .LT. 0)NAME=SYMBOL
        GO TO 10
C
C
C       END OF PROGRAM UNIT
C
 80     IST=IST+ISIZE-NEX
        RETURN
        END
```

```
      SUBROUTINE DATINF(INF,IDAT,ITRIP,IEOF)
C
C     SUBROUTINE FOR LOAD8
C     THE ROUTINE UNPACKS INFORMATION FROM BINARY FILES GENERATED
C     BY THE ASSEMBLER. IT READS FROM DAT 2.
C     AUTHOR LEIF ANDERSSON 1974-07-04.
C     REVISED LEIF ANDERSSON 1975-01-30
C
C     THE SUBROUTINE IS IMPURE IN THE SENSE THAT INTERNAL VARIAB-
C     LES ARE RETAINED BETWEEN CALLS.
C
C     INF-    INFORMATINON WORD
C     IDAT-   DATA WORD
C     ITRIP-  SET ITRIP=-1 EACH TIME A NEW FILE IS TO BE UNPACKED.
C             DO NOT CHANGE IT OTHERWISE.
C     IEOF-   RETURNED POSITIVE NORMALLY, 0 IF END OF FILE AND NEGATIVE
C             IF READ ERROR.
C
C     SUBROUTINES REQUIRED
C             READ
C
      DIMENSION IVECT(26)
      DATA LUN/2/
C
      IEOF=1
      IF(ITRIP .GE. 0)GO TO 10
C
      CALL READ( LUN,0,IVECT(1),26,NPAIRS,IEOF)
      ITRIP=0
      IF(IEOF .LE. 0)RETURN
      IN=1
      NTRIP=(NPAIRS-1)/2
C
 10   I=ITRIP*4+3
      II=I+IN
      IDAT=IVECT(II)
      GO TO(20,30,40),IN
C
 20   INF=IVECT(I)[0:5]
      IN=2
      RETURN
C
 30   INF=IVECT(I)[6:11]
      IN=3
      RETURN
C
 40   INF=IVECT(I)[12:17]
      IN=1
      ITRIP=ITRIP+1
      IF(ITRIP .EQ. NTRIP)ITRIP=-1
      RETURN
      END
```

```
      SUBROUTINE RADASC(D,A,B)
C     CONVERTS RADIX50 TO 5/7 ASCII
C     AUTHOR LEIF ANDERSSON 1974-07-24
C
C     D-      DOUBLE INTEGER CONTAINING A NAME IN RADIX50
C     A-      DOUBLE INTEGER RETURNED WITH THE FIRST
C             5 CHARACTERS
C     B-      DOUBLE INTEGER RETURNED WITH THE
C             LAST CHARACTER
C     UNUSED PARTS OF A AND B ARE SPACE FILLED.
C
C     SUBROUTINES REQUIRED
C             NONE
C
      DOUBLE INTEGER D,A,B
      DIMENSION IC(6)
C
C     SEPARATE THE FIRST AND SECOND WORD OF THE
C     DOUBLE INTEGER D
C
      ID1=D[1:17]
      ID2=D[18:35]
C
C     EXTRACT THE SIXBIT OCTAL CODE
C
      IC(6)=MOD(ID2,40)
      ID2=ID2/40
      IC(5)=MOD(ID2,40)
      IC(4)=ID2/40
      IC(3)=MOD(ID1,40)
      ID1=ID1/40
      IC(2)=MOD(ID1,40)
      IC(1)=ID1/40
C
C     CONVERT THE SPECIAL CODE TO ASCII
C
      DO 10 I=1,6
      IC1=IC(I)
      IF(IC1 .EQ. 0)IC(I)=□40
      IF(IC1 .GT. 0 .AND. IC1 .LE. □32)IC(I)=IC1+□100
      IF(IC1 .EQ. □33)IC(I)=□45
      IF(IC1 .EQ. □34)IC(I)=□56
      IF(IC1 .GE. □35 .AND. IC1 .LE. □46)IC(I)=IC1+□23
      IF(IC1 .EQ. □47)IC(I)=□43
 10   CONTINUE
C
C     PACK IN 5/7
C
      A[0:6]=IC(1)
      A[7:13]=IC(2)
      A[14:20]=IC(3)
      A[21:27]=IC(4)
      A[28:34]=IC(5)
      B[0:6]=IC(6)
      B[7:13]=□40
      B[14:20]=□40
      B[21:27]=□40
      B[28:34]=□40
      RETURN
      END
```

```
      SUBROUTINE ASCRAD(A,B,D)
C     SUVROUTINE FOR LOAD8
C     CONVERTS 5/7 ASCII TO RADIX50
C     AUTHOR LEIF ANDERSSON 1974-07-25
C
C     A -     DOUBLE INTEGER CONTAINING FIRST 5 CHARACTERS
C     B -     DOUBLE INTEGER CONTAINING SIXTH CHARACTER
C     D -     DOUBLE INTEGER RETURNED WITH RESULT. IF A OR B CONTAINS
C             AN ILLEGAL CHARACTER, D IS RETURNED -1
C     UNUSED PARTS OF A AND B MUST BE SPACE FILLED
C
C     SUBROUTINES REQUIRED
C             NONE
C
      DOUBLE INTEGER A,B,D
      DIMENSION IC(6)
C
C     UNPACK A AND B INTO IC
C
      IC(1)=A[0:6]
      IC(2)=A[7:13]
      IC(3)=A[14:20]
      IC(4)=A[21:27]
      IC(5)=A[28:34]
      IC(6)=B[0:6]
C
C     CONVERT IC TO THE SPECIAL SIXBIT CODE
C
      DO 10 I=1,6
      IC1=IC(I)
      IF(IC1 .EQ . ¤40)IC(I)=0
      IF(IC1 .EQ. ¤43)IC(I)=¤47
      IF(IC1 .EQ. ¤45)IC(I)=¤33
      IF(IC1 .EQ. ¤56)IC(I)=¤34
      IF(IC1 .GE. ¤60 .AND. IC1 .LE. ¤71)IC(I)=IC1-¤23
      IF(IC1 .GE. ¤101 .AND. IC1 .LE. ¤132)IC(I)=IC1-¤100
      IF(IC1 .EQ. IC(I))GO TO 99
 10   CONTINUE
C
C     PACK IN RADIX50
C
      IC1=((IC(1)*40)+IC(2))*40+IC(3)
      IC2=((IC(4)*40)+IC(5))*40+IC(6)
      D[0:17]=IC1
      D[18:35]=IC2
      RETURN
C
C     ILLEGAL CHARACTER
C
 99   D=-1
      RETURN
      END
```

```
      SUBROUTINE LPOUT(IDAT,IADR,IFL)
C     SUBROUTINE FOR LOAD8
C     AUTHOR LEIF ANDERSSON 1974-10-16
C     REVISED LEIF ANDERSSON 1975-04-01
C
C     THE ROUTINE WRITES OUTPUT DATA ON LP. IT IS IMPURE IN THE SENSE
C     THAT INTERNAL VARIABLES ARE RETAINED BETWEEN CALLS.
C     DAT +6 IS USED FOR THE LP.
C
C     IDAT -  OUTPUT DATA
C     IADR -  CURRENT ADDRESS
C     IFL -   FUNCTION INDICATOR: 1 INITIALIZE
C                                -1 TERMINATE, WRITE THE OUTPUT BUFFE
C             DO NOT CHANGE IFL EXCEPT ON THESE OCCASIONS
C
C     SUBROUTINES REQUIRED
C             NONE
      DIMENSION IBUF(8)
      IF (IFL)30,20,10
 10   DO 15 I=1,8
 15   IBUF(I)=0
      I=MOD(IADR,8)
      IFL=0
 20   I=I+1
      IBUF(I)=IDAT
      IF(I .NE. 8)RETURN
 30   IF(I .EQ. 0)RETURN
      IH=IADR/256
      IL=8*(MOD(IADR,256)/8)
      WRITE(6,100)IH,IL,(IBUF(J),J=1,I)
 100  FORMAT(1XO3,':',O3,8(3XO3))
      I=0
      IFL=1
      RETURN
      END
```

```
      SUBROUTINE PPOUT(IDAT,IFP)
C     SUBROUTINE FOR LOAD8
C     AUTHOR LEIF ANDERSSON 1974-10-16
C     REVISED LEIF ANDERSSON 1975-04-01
C
C     THE ROUTINE WRITES OUTPUT DATA ON PP. IT IS IMPURE IN THE
C     SENSE THAT INTERNAL VARIABLES ARE RETAINED BETWEEN CALLS.
C     DAT +7 IS USED FOR THE PP.
C
C     IDAT -  OUTPUT DATA
C     IFP -   FUNCTION INDICATOR: 1 INITIALIZE
C                                -1 TERMINATE. WRITE THE OUTPUT BUFFER
C             DO NOT CHANGE IFP EXCEPT ON THESE OCCASIONS.
C
C     SUBROUTINES REQUIRED
C             WRITE
C
      DIMENSION IBUF(53)
      IF(IFP)30,20,10
 10   IB=2
      IFP=0
 20   IB=IB+1
      IBUF(IB)=IDAT
      IF(IB .NE. 52)RETURN
 30   IF(IB .EQ. 0)RETURN
      IB=IB+1
      IBUF(IB)=0
      CALL WRITE(7,3,IBUF(1),IB)
      IFP=1
      IB=0
      RETURN
      END
```

```
      SUBROUTINE DKOUT(IDAT,IFD)
C     SUBROUTINE FOR LOAD8
C     AUTHOR LEIF ANDERSSON 1974-10-16
C     REVISED LEIF ANDERSSON 1975-04-01
C
C     WRITES OUTPUT DATA ON DISK. THE FILE MUST BE OPENED BEFORE
C     THE CALL. THE ROUTINE IS IMPURE IN THE SENSE THAT INTERNAL
C     VARIABLES ARE RETAINED BETWEEN CALLS.
C     DAT +1 IS USED FOR THE DISK OUTPUT.
C
C     IDAT -  OUTPUT DATA
C     IFD-    FUNCTION INDICATOR: 1 INITIALIZE
C                                -1 TERMINATE, WRITE THE OUTPUT BUFFER
C             DO NOT CHANGE IFD EXCEPT ON THESE OCCASIONS.
C
C     SUBROUTINES REQUIRED
C             NONE
C
      DIMENSION IBUF(8)
      IF(IFD)30,20,10
 10   I=0
      IFD=0
 20   I=I+1
      IBUF(I)=IDAT
      IF(I .NE. 8)RETURN
 30   IF(I .EQ. 0)RETURN
      WRITE(1,100)(IBUF(J),J=1,I)
 100  FORMAT(1X8(3XO3))
      IFD=1
      I=0
      RETURN
      END
```

```
      SUBROUTINE RADR(I,IAD,IND)
C     READS ADDRESS IN FORMAT <HIGH ADDRESS>:<LOW ADDRESS>
C     AUTHOR LEIF ANDERSSON 1975-06-30
C
C     I -      START AT CHARACTER NUMBER I
C     IAD -    RETURNED ADDRESS
C     IND -    INDICATES SUCCESS
C              1: ADDRESS FOUND
C              2: ADDRESS NOT FOUND, I IS UNCHANGED
C
C     SUBROUTINES REQUIRED
C              FAC
C
      LOGICAL NODIG
      DOUBLE INTEGER DID
      II=I-1
      IH=0
      IL=0
      NODIG=.TRUE.
      IND=2
 5    II=II+1
      CALL FAC(II,DID,IT)
      IF(IT .EQ. 3)GO TO 5
      IF(IT .NE. 2)RETURN
 10   CALL FAC(II,DID,IT)
      IF(IT .NE. 2)GO TO 20
      NODIG=.FALSE.
      IDIG=DID[3:6]
      IF(IDIG .GT. 7)RETURN
      IH=8*IH+IDIG
      II=II+1
      GO TO 10
 20   IF(NODIG .OR. DID .NE. ':')RETURN
      NODIG=.TRUE.
      II=II+1
 25   CALL FAC(II,DID,IT)
      IF(IT .NE. 2)GO TO 30
      NODIG=.FALSE.
      IDIG=DID[3:6]
      IF(IDIG .GT. 7)RETURN
      IL=8*IL+IDIG
      II=II+1
      GO TO 25
 30   IF(NODIG .OR. IL .GT. 255)RETURN
      IAD=IH*256+IL
      IND=1
      I=II
      RETURN
      END
```

```
      SUBROUTINE RID(I,DID,IND)
C
C     READS IDENTIFIER OR DELIMITER
C     AUTHOR LEIF ANDERSSON  1975-05-04
C
C     I -     START AT CHARACTER NUMBER I
C     DID -   RETURNED IDENTIFIER OR DELIMITER
C     IND -   INDICATES THE SUCCESS OF THE OPERATION
C             1: IDENTIFIER FOUND. RETURNED IN DID
C             2: IDENTIFIER FOUND. IT IS LONGER THAN FIVE CHARACTERS.
C                FIRST FIVE CHARACTERS ARE IN DID. I POINTS TO
C                THE NEXT CHARACTER.
C             3: DELIMITER FOUND. RETURNED IN DID.
C             4: THE FIRST CHARACTER OF THE FIELD WAS A DIGIT. I AND D
C                ARE UNCHANGED.
C             5: CARRIAGE RETURN
C             6: ALTMODE
C             7: UNRECOGNIZED CHARACTER OR I WAS NEGATIVE
C
C     ENTRRY RID2(I,DID,IND)
C
C     READS THE REMAINING PARTS OF LONG IDENTIFIERS
C
C     I -     START AT CHARACTER NUMBER I
C     DID -   RETURNED PART OF THE IDENTIFIER
C     IND -   INDICATES SUCCESS
C             1: IDENTIFIER TERMINATED
C             2: IDENTIFIER NOT TERMINATED, FIVE CHARACTERS IN DID
C     CAUTION
C     *******
C     RID2 SHOULD BE CALLED ONLY AFTER A CALL TO RID OR ANOTHER
C     CALL TO RID2. IF I POINTS TO ANYTHING BUT A LETTER OR A DIGIT
C     DID IS RETURNED BLANK,I IS UNCHANGED AND IND IS RETURNED 1
C
C     SUBROUTINES REQUIRED
C             FAC
C             PUTCHA
C
      DOUBLE INTEGER DID,CHAR
      DATA MAXCHA /5/
C
      IND=7
      ICHAR=0
      IF(I .LE. 0)RETURN
      GO TO 12
 10   I=I+1
 12   CALL FAC(I,CHAR,IND)
      GO TO(25,15,10,20,23,23,23),IND
C
C     NUMBER FOUND
C
 15   IND=4
      RETURN
C
C     DELIMITER FOUND
C
 20   DID =CHAR
      IND=3
      I=I+1
 23   RETURN
C
```

```
C        IDENTIFIER FOUND
C
 25      DID='      '
         IND=1
 30      ICHAR=ICHAR+1
         CALL PUTCHA(ICHAR,DID,CHAR)
         I=I+1
 35      CALL FAC(I,CHAR,IN1)
         IF(IN1 .GT. 2)RETURN
         IF(ICHAR .LT. MAXCHA)GO TO 30
         IND=2
         RETURN
C
         ENTRY RID2(I,DID,IND)
         DID='      '
         ICHAR=0
         IND=1
         GO TO 35
         END
```

```
      SUBROUTINE PUTCHA(I,A,CHAR)
C
C     PACKS CHARACTER INTO DOUBLEWORD
C     AUTHOR LEIF ANDERSSON 1975-05-04
C
C     I -     CHARACTER NUMBER I. VALUE 1 - 5.
C     A -     DOUBLE WORD TO RECEIVE CHARACTER.
C     CHAR -  CHARACTER TO BE INSERTED, FORMAT A1.
C
C     SUBROUTINES REQUIRED
C             NONE
C
      DOUBLE INTEGER A,CHAR
C
      ICHAR=CHAR[0:6]
      GO TO (10,20,30,40,50),I
 10   A[0:6]=ICHAR
      RETURN
 20   A[7:13]=ICHAR
      RETURN
 30   A[14:20]=ICHAR
      RETURN
 40   A[21:27]=ICHAR
      RETURN
 50   A[28:34]=ICHAR
      RETURN
      END
```

```
/         SUBROUTINE INIT(LUN,IDIR)
/
/         PERFORMS INIT ON A LOGICAL UNIT.
/         AUTHOR LEIF ANDERSSON 1974-10-23
/
/         LUN -    LOGICAL UNIT NUMBER
/         IDIR -   DIRECTION:      0 - INPUT
/                                  1 - OUTPUT
/
/         SUBROUTINES REQUIRED
/                  NONE
/
/         EDIT #001
/
          .GLOBL INIT,.DA
INIT      0
          JMS* .DA
          JMP .+3
LUN       0
IDIR      0
          LAC* LUN
          AND (777
          DAC CALINS
          CLA!IAC
          AND* IDIR
          SWHA
          TAD CALINS
          DAC CALINS
CALINS    0
          1
          CALINS
          0
/
          JMP* INIT
          .END
```

```
/       SURROUTINE WRITE(LUN,MODE,IBUF(*),ISIZE)
/
/       WRITES ON ANY UNIT IN ANY MODE.
/       AUTHOR LEIF ANDERSSON 1974-10-23
/
/       IN ALL DATA MODES EXCEPT DUMP HEADER WORD 0 IS COMPUTED AND
/       INSEPTED. NO CHECK IS MADE ON THE LEGALITY OF THE NUMBERS
/       PASSED TO THE ROUTINE.
/
/       LUN -   LOGIVAL UNIT NUMBER
/       MODE -  DATA MODE:      0 -IOPS BINARY
/                               1 - IMAGE BINARY
/                               2 - IOPS ASCII
/                               3 - IMAGE ALPHANUMERIC
/                               4 - DUMP
/       IBUF -  LINE BUFFER ADDRESS
/       ISIZE - NUMBER OF WORDS TO BE WRITTEN INCLUDING HEADER WORD PAIR
/
/       SUBROUTINES REQUIRED
/               NONE
/
/       EDIT #001
/
        .GLOBL WRITE,.DA
WRITE   0
        JMS* .DA
        JMP .+5
LUN     0
MODE    0
IBUF    0
ISIZE   0
/MASK LUN AND MODE
        LAC* LUN
        AND (777
        DAC LUN1
        LAC* MODE
        AND (7
        DAC CALINS
/CHECK IF DUMP MODE
        LAW -4
        TAD CALINS
        SNA
        JMP BCAL
/COMPUTE AND INSERT HEADER WORD 0.
        LAC* ISIZE
        AND (777
        RCR
        SWHA
        DAC* IBUF
/BUILD THE CAL BLOCK FOR WRITE
BCAL    LAC CALINS
        SWHA
        TAD LUN1
        DAC CALINS
/
        LAC IBUF
        DAC BUF
/
        LAC* ISIZE
        TCA
        DAC SIZE
```

```
/CAL BLOCK FOR WRITE
CALINS  0
        11
BUF     0
SIZE    0
/CAL BLOCK FOR WAIT
LUN1    0
        12
/
        JMP* WRITE
        .END
```

```
/       SUBROUTINE READ(LUN,MODE,IBUF(*),ISIZE,NPAIRS,IVAL)
/
/       READS FROM ANY UNIT IN ANY MODE.
/       AUTHOR LEIF ANDERSSON 1974-06-26
/
/       LUN -    LOGICAL UNIT NUMBER
/       MODE -   DATA MODE:        0=IOPS BINARY
/                                  1=IMAGE BINARY
/                                  2=IOPS ASCII
/                                  3=IMAGE ALPHANUMERIC
/                                  4=DUMP
/       IBUF -   LINE BUFFER ADDRESS
/       ISIZE -  LINE BUFFER SIZE
/       NPAIRS-   WORD PAIR COUNT
/       IVAL -   VALIDITY. IF POSITIVE: NORMAL EXIT
/                IF ZERO: END OF FILE OR END OF MEDIUM
/                IF NEGATIVE: READ ERROR: -1=PARITY
/                                         -2=CHECKSUM
/                                         -3=BUFFER OVERFLOW
/
/
/       SUBROUTINES REQUIRED
/               NONE
/
/       EDIT #002
/
        .GLOBL READ,.DA
READ    0
        JMS* .DA
        JMP .+7
LUN     0
MODE    0
IBUF    0
ISIZE   0
NPAIRS  0
IVAL    0
/
/BUILD CAL BLOCK FOR READ
/
        LAC* MODE
        SWHA
        TAD* LUN
        DAC CALINS
        LAC IBUF
        DAC BUF
        LAC* ISIZE
        TCA
        DAC SIZE
/
/CAL BLOCK FO READ
/
CALINS  CAL 0
        10
BUF     0
SIZE    0
/
/BUILD CAL BLOCK FOR WAIT
/
        LAC* LUN
        DAC WAIT
WAIT    CAL 0
        12
```

```
/
/GET WORD PAIR COUNT
/
        LAC* IBUF
        SWHA
        AND (377
        DAC* NPAIRS
/
/CHECK VALIDITY
/
        LAC* IBUF
        AND (77
        LRSS 4
        SNA
        JMP OK
        TCA
        DAC* IVAL
        JMP* READ
/
/CHECK EOF OR EOM
/
OK      LAC* IBUF
        AND (17
        SAD (5
        JMP EOFM
        SAD (6
        JMP EOFM
        CLA!IAC /INITIALIZE IVAL FOR NORMAL EXIT
        DAC* IVAL
        JMP* READ
EOFM    DZM* IVAL
        JMP* READ
        .END
```

```
C       PROGRAM MAIN40
C
C       MAIN PROGRAM FOR LOAD40
C       AUTHOR LEIF ANDERSSON 1974-11-13
C       REVISED JANUSZ MISZCZUK 1975-09-03
C
C       SUBROUTINES REQUIRED
C               REL40
C               SNAM
C               COMND
C               MATCH
C               GLOBLS
C               DATINF
C               RADASC
C               ASCRAD
C               LPOUT
C               PPOUT
C               DKOUT
C               RADR
C               RID
C               PUTCHA
C               RLINE
C               FAC
C               INIT
C               READ
C               WRITE
C
        DOUBLE INTEGER FILNAM,GLOB,EXTREF,NAME,NOMTCH,FN,DNAM,
        1                    BUFF,LIBR,A,B,ALTM
        DIMENSION FILNAM(25),GLOB(100),IADR(100),EXTREF(100),
        1            NAME(100),IST(100),NEX(100),NOMTCH(100),FN(2),
        2            DNAM(2),BUFF(16)
        COMMON FILNAM,GLOB,IADR,EXTREF,NOMTCH,NAME,IST,NEX
        DATA FN(2)/' BIN'/,
        1    DNAM(2)/' ABS'/,
        2    LIBR/'NONE'/,
        3    ALTM/¤D764000000000/
C
C       GET COMMANDS
C
 10     WRITE(9,100)
 100    FORMAT(1X'LOAD40 V1A')
        IGLOB=0
        IEX=0
        CALL COMND(LIBR,GLOB,IADR,IGLOB,IST1,FILNAM,NFIL,DNAM(1),IND)
        IG1=IGLOB+1
        IST(1)=IST1
        IU=1
        NEX(1)=1
C
C       GET GLOBAL SYMBOLS AND EXTERNAL REFERENCES
C
        DO 20 I=1,NFIL
        FN(1)=FILNAM(I)
        CALL SEEK(2,FN)
 15     CALL GLOBLS(IST1,NAME(IU),GLOB,IADR,IGLOB,EXTREF,IEX,IEOF)
        IF(IEOF)99,20,16
 16     IU=IU+1
        NEX(IU)=IEX+1
        IST(IU)=IST1
        GO TO 15
```

```
 20     CALL CLOSE(2)
        IG2=IGLOB
C
C       GET THE GLOBALS FOR THE LIBRARY PROGRAMS IF ANY ARE NEEDED
C
        CALL MATCH(EXTREF,IEX,GLOB,IGLOB,NOMTCH,IN,M)
        IF(IN .EQ. 0)GO TO 50
        IF(LIBR .EQ. 'NONE')GO TO 40
        NFIL=NFIL+1
        FILNAM(NFIL)=LIBR
        FN(1)=LIBR
        CALL SEEK(2,FN)
C
 25     IG=IGLOB
        IE=IEX
        IST1=IST(IU)
 26     CALL GLOBLS(IST1,NAME(IU),GLOB,IADR,IG,EXTREF,IE,IEOF)
        IF(IEOF)99,29,27
 27     IN1=IN
        CALL MATCH(NOMTCH,IN1,GLOB,IG,NOMTCH,IN,M)
        IF(M .EQ. 0)GO TO 25
        CALL MATCH(EXTREF,IE,GLOB,IG,NOMTCH,IN,M)
        IU=IU+1
        IEX=IE
        IGLOB=IG
        NEX(IU)=IEX+1
        IST(IU)=IST1
        IF(IN .NE. 0)GO TO 26
 29     CALL CLOSE(2)
        IG2=IGLOB
        IF(IN .EQ. 0)GO TO 50
C
C       GET THE UNRESOLVED GLOBALS IF ANY
C
 40     WRITE(9,110)
 110    FORMAT(1X'UNRESOLVED GLOBALS')
        DO 45 II=1,IN
        CALL RADASC(NOMTCH(II),A,B)
        WRITE(9,120)A,B,ALTM
 120    FORMAT(1X,A5,A5,A1)
 43     CALL RLINE(9,8,BUFF,0)
        I=1
        CALL RADR(I,IAD,IA)
        IF(IA .EQ. 1)GO TO 44
        WRITE(9,130)
 130    FORMAT(1X'SYNTAX ERROR, RETYPE ADDRESS.')
        GO TO 43
 44     IGLOB=IGLOB+1
        GLOB(IGLOB)=NOMTCH(II)
 45     IADR(IGLOB)=IAD
C
C       WRITE LOAD MAP AND GLOBAL SYMBOL TABLE
C
 50     WRITE(6,200)
 200    FORMAT(1X'FILE NAMES')
        IF(LIBR .NE. 'NONE') NFIL=NFIL-1
        DO 55 I=1,NFIL
 55     WRITE(6,210)FILNAM(I)
 210    FORMAT(5XA5)
C
        WRITE(6,220)LIBR
```

```
 220    FORMAT('0'/1X'LIBRARY:  ',A5)
C
        IL1=MOD(IST(1),256)
        IH1=IST(1)/256
        IST1=IST(IU)-1
        IL2=MOD(IST1,256)
        IH2=IST1/256
        WRITE(6,230)IH1,IL1,IH2,IL2
 230    FORMAT('0'/1X'MEMORY REQUIRED:  ',O3,':',O3,' - ',
        1       O3,':',O3)
C
        WRITE(6,240)
 240    FORMAT('0'/1X'LOAD MAP')
        IU=IU-1
        DO 60 I=1,IU
        CALL RADASC(NAME(I),A,B)
        IL=MOD(IST(I),256)
        IH=IST(I)/256
 60     WRITE(6,250)A,B,IH,IL
 250    FORMAT(5XA5,A5,O3,':',O3)
C
        IF(IG1 .GT. IG2)GO TO 70
        WRITE(6,260)
 260    FORMAT('0'/1X'GLOBAL SYMBOL TABLE')
        DO 65 I=IG1,IG2
        CALL RADASC(GLOB(I),A,B)
        IL=MOD(IADR(I),256)
        IH=IADR(I)/256
 65     WRITE(6,250)A,B,IH,IL
C
 70     WRITE(6,270)
 270    FORMAT('1')
C
C       START RELOCATING
C
        IF(IND .NE. 2)GO TO 73
        CALL ENTER(1,DNAM)
        WRITE(1,230)IH1,IL1,IH2,IL2
 73     IF(IND .NE. 1)GO TO 75
        CALL INIT(7,1)
        IFP=1
        CALL PPOUT(□377,IFP)
C
 75     IV=1
        IFIL=0
        ICUR=IST(1)
        ICHECK=0
        DO 80 I=1,IU
        II=I+1
        CALL SNAM(FILNAM,IFIL,NAME(I),ITRIP,IERR)
        IF(IERR .LT. 0)GO TO 99
        CALL REL40(ICUR,IST(II),GLOB,IADR,EXTREF,NEX(I),
        1          IND,IV,ICHECK,ITRIP,IFP)
        IF(IV .EQ. -1)GO TO 99
        IF(IV .EQ. -2)GO TO 98
 80     CONTINUE
C
        CALL CLOSE(2)
        IFL=-1
        IFD=-1
        CALL LPOUT(I,ICUR,IFL)
```

```
      IF(IND .NE. 2)GO TO 85
      CALL DKOUT(I,IFD)
      CALL CLOSE(1)
 85   IF(IND .NE. 1)GO TO 90
      CALL PPOUT(ICHECK,IFP)
      IFP=-1
      CALL PPOUT(I,IFP)
      CALL CLOSE(7)
 90   WRITE(6,270)
      GO TO 10
C
      READ(2)I
      WRITE(7)I
 99   WRITE(9,300)
 300  FORMAT(1X'READ ERROR')
 98   WRITE(9,310)
 310  FORMAT(1X'CODE ERROR')
      GO TO 10
      END
```

```
      SUBROUTINE REL40(ICUR,ISN,GLOB,IADR,EXTREF,NEX,
     1                 IND,IV,ICHECK,ITRIP,IFP)
C
C     RELOCATION SUBROUTINE FOR LOAD40
C     AUTHOR JANUSZ MISZCZUK 1975-09-03
C
C     THE INPUT AND OUTPUT FILE MUST BE OPENED BEFORE THE CALL
C     AND INIT MUST BE DONE ON THE PP
C
C     ICUR -   CURRENT RELOCATION ADDRESS
C     ISN -    START ADDRESS OF NEXT PROGRAM UNIT
C     GLOB -   DOUBLE INTEGER VECTOR WITH GLOBAL NAMES
C     IADR -   VECTOR WITH THE ADDRESSES OF THE GLOBALS
C     EXTREF-  DOUBLE INTEGER VECTOR WITH THE NAMES OF EXTERNAL REFS
C     NEX -    INDEX OF THE FIRST EXTERNAL REFERENCE FOR THE CURRENT
C              PROGRAM UNIT
C     IND -    OUTPUT INDICATOR          0: LP ONLY
C                                        1: PP+LP
C                                        2: DK+LP
C     IV -     SET IV=1 ON THE FIRST CALL. IV IS RETURNED 0
C              NORMALLY, -1 IF READ ERROR AND -2 IF CODE ERROR
C     ICHECK-  CHECKSUM
C     ITRIP-   PARAMETER FOR DATINF. DO NOT CHANGE IT.
C     IFP -    THIS IS THE PARAMETER IFP OF PPOUT.
C
C     SUBROUTINES REQUIRED
C              DATINF
C              (READ)
C              LPOUT
C              DKOUT
C              PPOUT
C              (WRITE)
C
      DOUBLE INTEGER GLOB,EXTREF,NAME
      DIMENSION GLOB(1),EXTREF(1),IADR(1)
C
      IST=ICUR
      IF(IV .NE. 1)GO TO 10
      IV=0
      IFL=1
      IFD=1
C
 10   CALL DATINF(NINF,NDAT,ITRIP,IEOF)
      IF(IEOF)99,99,15
 15   IF(NINF .EQ. 4)GO TO 20
      IF(NINF .EQ. 5)GO TO 98
      GO TO 10
 20   IDAT=NDAT[10:17]
 30   ICUR1=ICUR+1
C
C
      IF(ICUR1 .EQ. ISN) GO TO 130
 40   CALL DATINF(NINF,NDAT,ITRIP,IEOF)
      IF(IEOF)99,99,45
 45   IF(NINF .EQ. 4) GO TO 50
      IF(NINF .EQ. 5) GO TO 60
      GO TO 40
C
C     ABSOLUTE INSTRUCTION OR CONSTANT
C
 50   IDAT1=NDAT[10:17]
```

```
C
C
      IF(IND .EQ. 1)CALL PPOUT(IDAT,IFP)
      IF(IND .EQ. 2)CALL DKOUT(IDAT,IFD)
      CALL LPOUT(IDAT,ICUR,IFL)
C
      ICHECK=ICHECK+IDAT
      ICUR=ICUR+1
      IDAT=IDAT1
      GO TO 30
C
C     RELOCATABLE ADDRESS
C
 60   IA=NDAT+IST
      IF(IA .LT. ISN) GO TO 80
C
C     EXTERNAL REFERENCE
C
      IA=IA-ISN+NEX
      NAME=EXTREF(IA)
      I=0
 70   I=I+1
      IF(NAME .NE. GLOB(I)) GO TO 70
      IA=IADR(I)
C
C     OUTPUT SECTION
C
 80   ID=IDAT/16
      IC1=MOD(ICUR1,256)
      MSIC1=ICUR1/256
      IA1=IA/256
      IA2=MOD(IA,256)
      IF(ID .NE. 1 .AND. ID .NE. 7) GO TO 100
      IF(IC1 .EQ. 255) GO TO 90
      IF (IA1-MSIC1 .EQ. 0) GO TO 110
      WRITE(9,160) MSIC1,IC1,IDAT,IA1,IA2
      GO TO 110
 90   IF(IA1-MSIC1 .EQ. 1) GO TO 110
      WRITE(9,180) MSIC1,IDAT,IA1,IA2
C
C
 100  IF(IDAT .EQ. ¤100 .OR. IDAT .EQ. ¤120)IDAT=IDAT+IA/256
 110  IDAT1=MOD(IA,256)
C
C     FIRST WORD OF TWO WORD INSTRUCTION , ABSOLUTE
C     INSTRUCTION OR CONSTANT
C
 120  IF(IND .EQ. 1)CALL PPOUT(IDAT,IFP)
      IF(IND .EQ. 2)CALL DKOUT(IDAT,IFP)
      CALL LPOUT(IDAT,ICUR,IFL)
C
      ICHECK=ICHECK+IDAT
      ICUR=ICUR+1
C
C     RELOCATABLE ADDRESS OR THE LAST WORD OF
C     SUBROUTINE
C
      IDAT=IDAT1
 130  IF(IND .EQ. 1)CALL PPOUT(IDAT,IFP)
      IF(IND .EQ. 2)CALL DKOUT(IDAT,IFP)
      CALL LPOUT(IDAT,ICUR,IFL)
```

```
      ICHECK=ICHECK+IDAT
      ICUR=ICUR+1
      IF(ICUR .EQ. ISN) RETURN
      GO TO 10
C
C     CODE ERROR
C
 98   IV=-2
      RETURN
C
C     READ ERROR
C
 99   IV=-1
      RETURN
C
 160  FORMAT(1X'ADDRESS ERROR FOR JCN OR ISZ AT  '
     1       ,O3,':',O3,'  CODE=',O3/1X'DESTINATION',
     2       ' AT  ',O3,':',O3)
 180  FORMAT(1X'ADDRESS ERROR FOR JCN OR ISZ AT  '
     1       ,O3,':377  CODE=',O3/1X'DESTINATION AT '
     2       ,O3,':',O3)
      END
```