**Source Listings to Z80 Cross Assembler and Linking Loader included as part of report.**

**Z80 microprocessor, assembler, linker, loader, PASCAL**

The Z80 Cross Assembler described by this document is a Zilog-standard cross assembler written in PASCAL. It supports all the Zilog-standard mnemonics, but its pseudo-ops are unique to this assembler (to some extent). This assembler produces relocatable code which may be later linked and loaded by its companion linking loader.

This cross assembler was customized for application with the ARIAN II operating system. It is designed to be run on a CDC 6000 or CYBER class.
20. (cont)

machine. The intended operating environment is in a timesharing mode on the host computer with a microcomputer linked to the host for communications and, specifically, downloading.

- End of Information -
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Use of the Assembler</td>
<td>2</td>
</tr>
<tr>
<td>2.1 Pseudo-Ops</td>
<td>2</td>
</tr>
<tr>
<td>2.2 Addressing Modes</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Expression Syntax</td>
<td>5</td>
</tr>
<tr>
<td>2.4 Invoking the Assembler</td>
<td>5</td>
</tr>
<tr>
<td>3. Use of the Linker</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Library Files</td>
<td>7</td>
</tr>
<tr>
<td>3.2 Invoking the Linker</td>
<td>7</td>
</tr>
<tr>
<td>4. Appendices</td>
<td>9</td>
</tr>
<tr>
<td>A. Summary of Z80 Mnemonics</td>
<td>9</td>
</tr>
<tr>
<td>B. Relocatable Object Format</td>
<td>13</td>
</tr>
<tr>
<td>C. Assembler Error Messages</td>
<td>15</td>
</tr>
<tr>
<td>D. References</td>
<td>16</td>
</tr>
<tr>
<td>5. Source Code Listings</td>
<td>17</td>
</tr>
<tr>
<td>ZLDR</td>
<td></td>
</tr>
<tr>
<td>Z80ASM</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 1

Introduction

The Z80 Cross Assembler was written for a number of reasons. Among them are: (1) the need of an assembler customized for the Project ARIES applications, (2) the inefficiency of the MAC80 cross assembler, and (3) the lack of linking loader facilities for Z80 cross software. This assembler provides the basis for a linking loader environment by establishing a relocatable object format.

This assembler is designed to fully support the Zilog-standard mnemonics for the Z80 op codes. All Zilog-standard mnemonics are implemented by the assembler. The pseudo-ops, however, are mainly unique to this assembler.

The Z80 Cross Assembler, hereafter referred to as ASMZ80, produces three files as output. The first file is the output file to the user's terminal or batch job stream. Messages are sent to this file by the assembler. The second file is the assembler listing file. It is in a paged format suitable for line printer listing. Finally, the third file is the object file. It is in the Project ARIES object format.
The assembler is very easy to use if one is familiar with timesharing on the CDC 6500/6600 under INTERCOM. If not, see ARRADCOM MISD consultants at Picatinny Arsenal on how to get started. Basically, one will need to put his Z80 program into a text file by using an editor. When this has been done, the user may then save his program as a local file and use the assembler. Two local files are generated by the assembler, a program listing file and an object file. This object file is called a paper-tape object file because it is suitable for punching on paper tape. The paper tape format also makes the file suitable for transmission over telephone lines to a microcomputer.

2.1 Pseudo-ops

The pseudo-ops are one of the features that make one assembler different from another. For this reason, the pseudo-ops of ASMZ80 are listed in some detail.

DB - define byte. The operand(s) of this pseudo-op are evaluated and emitted one byte per operand. Strings (enclosed in single quotes) are an exception to this, with one byte being emitted for each character in the string (not counting the beginning and ending single quotes).

DEFB -- define byte. This pseudo-op was included to help maintain compatibility with the Zilog-standard pseudo-ops. The DEFB function is a subfunction of DB, and DEFB in Z80ASM is implemented exactly as DB.

DEFM -- define message. Same as DEFB.

DEFS -- define storage. This pseudo-op was included to help maintain compatibility with the Zilog-standard pseudo-ops. The DEFS function is a subfunction of DS, and DEFS in Z80ASM is implemented exactly as DS.

DEFW -- define word. This pseudo-op was included to help maintain compatibility with the Zilog-standard pseudo-ops. The DEFW function is a subfunction of DW, and DEFW in Z80ASM is implemented exactly as DW.
DISPLAY - display the value of an expression on the user's terminal. This is useful when the user wishes to know a particular value computed by the assembler without having to scan through the listing. For example, a typical use of DISPLAY would be to display the range of an assembly.

DS - define storage. The operand field is evaluated, and that many bytes of memory are reserved starting with the current value of the memory location counter.

DW - define word. The operand(s) of this pseudo-op are evaluated and emitted one word (two bytes) per operand.

END - end of program. This pseudo-op signifies that there is no more program code in the source file.

EQU - this evaluates the operand field and assigns the value to the label given at the beginning of the line. Absence of a label will be noted as an error. Note that equates with register names will allow the user to employ the new symbols as he would employ the corresponding register name.

EXT - defines a symbol to be external. This will enable later reference to these symbols by the linking loader at load time. The EXT pseudo-op must be present in both the source in which the symbol is defined and the source in which the symbol is referenced. The label associated with EXT, then, is not processed as a normal label; it appears twice in the definition source.

HEADER - place a header at the top of the following pages. This header is located under the title generated by the TITLE pseudo-op. The string in the operand field (enclosed in single quotes) appears as the header.

LIST - turns on the program listing (generation of the listing file). This is the default.

MESSAGE - display a message on the user's terminal. This command is used to send the string in the operand field to the user's terminal. It may be used in conjunction with the DISPLAY pseudo-op to print a message along with the value displayed. The string in the operand field (enclosed in single quotes) appears as the message.

NOLIST - turns off program listing. The advantage of this pseudo-op is a slight increase in the efficiency of the assembler and lower printing costs.

NOSYM - inhibits the generation of a symbol table in the program listing. The generation of a symbol table is the default.

ONLY8080 - sets a switch which will raise an error if an instruction which will not execute on the 8080 is encountered. This option is disengaged as default.

ORG - sets the memory location counter of the assembler to the value of the expression in the operand field. If a label is present, it is also given that value. Secondly, this pseudo-op sets the address mode to absolute. The absolute addressing mode is the default. If an ORG is not specified, assembly will start with zero as the value of the memory location counter.

PAGE - forces a page eject. This will help in making listings more readable.

REL - sets the address mode to relative. No operand is required.
2.2 Addressing Modes

There are two addressing modes supported by this assembler — absolute and relative. If an entire assembly is run in absolute mode, then the resulting object module will be completely compatible with the ARIES/MUMS object format. To put the assembler in absolute mode, one simply uses the ORG pseudo-op, supplying the starting address of the memory location counter as the operand.
The relative addressing mode is more flexible. Using this mode, object libraries can be built using relocatable routines. The only problem encountered in relocation is the use of word-valued (two byte) addresses within the assembled program. In absolute mode, these addresses are indistinguishable from any pair of bytes, so relocation is impossible. In relative mode, these addresses are prefixed by the letter 'R' to signify a relative address. A two-byte value follows the 'R'; this is the offset from the beginning of the current relocatable module. Thus, when loading, this value is added to the value of the memory address at which the current code segment is to reside. Relative mode is entered by using the REL pseudo-op, which needs no operand. The default address mode of ASMZ80 is absolute.

Note that one can switch back and forth between absolute and relative addressing modes, referencing absolute symbols in relative code and relative symbols in absolute code. It is the responsibility of the programmer to be sure he is not overlaying some of his own code at load time if he chooses to change modes like this.

External symbols are provided to allow the programmer to use previously-assembled routines by name, rather than by the awkward method of using equates in which the programmer does the memory mapping. A symbol is declared external by the EXT pseudo-op, with the symbol at the beginning of the line being the label referenced. These externals can be either absolute or relative and must be resolved by the linker at load time.

2.3 Expression Syntax

Expressions analyzed by the assembler must be rather simple in form. Since parentheses are used for indirection, no effort was made to implement them in expressions. Hence, no nesting of expressions is permitted, and evaluation is done strictly from left to right with operator precedence. Also, no blanks are accepted within an expression, and addressing modes cannot be mixed freely. External reference symbols can't be used in expressions, and relative symbols can only be operated on by plus (+) and minus (-). The following operators are recognized by the assembler:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Priority</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>4</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>4</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>4</td>
<td>Modulo</td>
</tr>
<tr>
<td>+</td>
<td>3</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>3</td>
<td>Subtraction</td>
</tr>
<tr>
<td>&amp;</td>
<td>2</td>
<td>Logical AND</td>
</tr>
<tr>
<td>!</td>
<td>1</td>
<td>Logical OR</td>
</tr>
</tbody>
</table>

2.4 Invoking the Assembler

Using the assembler is very simple once one has written the text file. Use
of the assembler is done in two steps: (1) initialization, which is done only once during a timesharing session, and (2) invocation of the assembler itself. It is usually a good idea to initialize the user's timesharing environment immediately after logging in to the system. This is done by issuing the following INTERCOM commands:

```intercom
ATTACH,P1,PROFIL,ID=XXXXXX
REWIND,P1
COPY,P1,PROFIL
RETURN,P1
RFL,64000
ETL,200
CONNECT,INPUT,OUTPUT
```

Once this has been done, the user is ready to perform his normal timesharing activities. When the user's program source file is ready to be assembled, the user executes ASMZ80 by issuing the following INTERCOM command:

```intercom
BEGIN,ASSM,,SOURCE,LISTING,OBJECT,OBJECT2
```

SOURCE is the user's assembler language source file, LISTING is the assembler's output listing file, OBJECT is the assembler's INTEL-format output object file, and OBJECT2 is the assembler's ARIES-format output object file. All names are optional, the files Z8OSRC, Z8OLST, Z80OBJ1, and Z80OBJ2 being used if the replacement is not indicated. All files are rewound before and after using them.

The assembler will run in a field length of less than 20000B if the program is not large (less than 500 bytes), and has never (yet) run out of space with a field length of 64000B.

At the end of the assembly, the assembler will tell the user how many errors were detected. Errors are indicated in the output listing by a hash character ('#') in one of the first four columns of the error line. Hence, the user may use an editor to find the error lines by searching for this character in the first four columns. Also in the listing are the error message codes, located between the object code and actual source line (the actual source line is preceded by a colon, ':').

An error-listing procedure is available to the user through the PROFIL. This is invoked by

```intercom
BEGIN,ERRORS,,LISTING
```

where LISTING is the listing file generated by the assembly. This procedure will list all error lines on the user's terminal for his review.
CHAPTER 3
Use of the Linker

One of the functions of ASMZ80 is to allow the user to maintain object code libraries, removing the need to assemble all the routines he needs every time. To this goal, he must declare the names of his routines to be external when they are assembled and put the relocatable object into his library.

3.1 Library Files

The library file format is very simple, and it will allow the use of basic utilities to construct it. It is a segmented file of text, with each segment containing exactly one object module. For example, to add a newly-assembled object file, OBJ, to the library file, LIB, the user need only type:

REWIND,LIB
COPY,LIB,NEWLIB
REWIND,OBJ
COPY,OBJ,NEWLIB
REWIND,NEWLIB
REWIND,LIB
COPY,NEWLIB,LIB

This will add the object file to the library file, LIB.

3.2 Invoking the Linker

The procedure 'LINK' is used to run the linker. To invoke the linker, type:

BEGIN, LINK,, SOURCE, LIBRARY, OBJECT, OBJECT1, MESSAGE

where SOURCE is the main object module, LIBRARY is the object library file, OBJECT is the output object file in INTEL hex format, OBJECT1 is the output
object file in ARIES-format, and MESSAGE is the output message file produced by the linker which holds any diagnostics and a load map. SOURCE and LIBRARY must be relocatable object files in ARIES format.

When the linker starts running, the user will type a linker directive. The linker directives are:

R -- link and relocate. This tells the linker to leave the resolved object module in relocatable format.

A <address> -- link and generate absolute code. This tells the linker to output the object in absolute format, with the hex string specifying the start address. This command contains no blanks, and takes the form of 'AHHHH', where 'H' represents any valid hexadecimal digit.

Any errors other than multiple symbol definition are considered to be catastrophic and will abnormally terminate the linker.
Appendix A

Summary of Z80 Mnemonics

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Format</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>A, Q8</td>
<td>CCF</td>
</tr>
<tr>
<td>ADC</td>
<td>HL, R16A</td>
<td>Q8</td>
</tr>
<tr>
<td>ADD</td>
<td>A, Q8</td>
<td>CP</td>
</tr>
<tr>
<td>ADD</td>
<td>HL, R16A</td>
<td>CPD</td>
</tr>
<tr>
<td>ADD</td>
<td>IX, R16B</td>
<td>CPDR</td>
</tr>
<tr>
<td>ADD</td>
<td>IY, R16C</td>
<td>CPI</td>
</tr>
<tr>
<td>AND</td>
<td>Q8</td>
<td>CPL</td>
</tr>
<tr>
<td>BIT</td>
<td>N3, Q8A</td>
<td>DAA</td>
</tr>
<tr>
<td>CALL</td>
<td>N16</td>
<td>DEC</td>
</tr>
<tr>
<td>CALL</td>
<td>CF, N16</td>
<td>DEC</td>
</tr>
</tbody>
</table>

Page 9
### Summary of Z80 Mnemonics, Con't

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI</td>
<td>(IY)</td>
</tr>
<tr>
<td>DJNZ</td>
<td>JP N16</td>
</tr>
<tr>
<td></td>
<td>JP CF,N16</td>
</tr>
<tr>
<td>EI</td>
<td>JR E</td>
</tr>
<tr>
<td></td>
<td>JR CFA,E</td>
</tr>
<tr>
<td>EX</td>
<td>LD RB,Q8</td>
</tr>
<tr>
<td>EX</td>
<td>LD A,(N16)</td>
</tr>
<tr>
<td>EX</td>
<td>LD (N16),A</td>
</tr>
<tr>
<td>EX</td>
<td>LD R16D,(N16)</td>
</tr>
<tr>
<td>EX</td>
<td>LD (N16),R16D</td>
</tr>
<tr>
<td>EXX</td>
<td>LD R16D,N16</td>
</tr>
<tr>
<td>HALT</td>
<td>LD SP,HL</td>
</tr>
<tr>
<td>IM</td>
<td>LD SP,IX</td>
</tr>
<tr>
<td>IM</td>
<td>LD SP,IY</td>
</tr>
<tr>
<td>IM</td>
<td>LD A,R</td>
</tr>
<tr>
<td>IM</td>
<td>LD A,I</td>
</tr>
<tr>
<td>IN</td>
<td>LD R,A</td>
</tr>
<tr>
<td>IN</td>
<td>LD I,A</td>
</tr>
<tr>
<td>INC</td>
<td>LD A,(BC)</td>
</tr>
<tr>
<td>INC</td>
<td>LD A,(DE)</td>
</tr>
<tr>
<td>IND</td>
<td>LD (BC),A</td>
</tr>
<tr>
<td>IND</td>
<td>LD (DE),A</td>
</tr>
<tr>
<td>INDIR</td>
<td>LDD</td>
</tr>
<tr>
<td>INDIR</td>
<td>LDDR</td>
</tr>
<tr>
<td>INIT</td>
<td>LDI</td>
</tr>
<tr>
<td>INIR</td>
<td>LDIR</td>
</tr>
<tr>
<td>JP</td>
<td>NEQ</td>
</tr>
<tr>
<td>JP</td>
<td>NOP</td>
</tr>
</tbody>
</table>
# Summary of Z80 Mnemonics, Con't

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Notes</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>Q8</td>
<td>RLCA</td>
<td></td>
</tr>
<tr>
<td>OITDR</td>
<td></td>
<td>RLD</td>
<td></td>
</tr>
<tr>
<td>OITIR</td>
<td></td>
<td>RR</td>
<td>Q8A</td>
</tr>
<tr>
<td>OUTD</td>
<td></td>
<td>RRA</td>
<td></td>
</tr>
<tr>
<td>OUTI</td>
<td></td>
<td>RRC</td>
<td>Q8A</td>
</tr>
<tr>
<td>OUT (N8),A</td>
<td></td>
<td>RRCA</td>
<td></td>
</tr>
<tr>
<td>OUT (C),R8</td>
<td></td>
<td>RRD</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td>R16E</td>
<td>RRD</td>
<td></td>
</tr>
<tr>
<td>PUSH</td>
<td>R16E</td>
<td>RST</td>
<td>N3#8</td>
</tr>
<tr>
<td>RES</td>
<td>N3,Q8A</td>
<td>SCF</td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td>CF</td>
<td>SET</td>
<td>N3,Q8A</td>
</tr>
<tr>
<td>RETI</td>
<td></td>
<td>SRA</td>
<td>Q8A</td>
</tr>
<tr>
<td>RETN</td>
<td></td>
<td>SRL</td>
<td>Q8A</td>
</tr>
<tr>
<td>RL</td>
<td>Q8A</td>
<td>SUB</td>
<td>Q8</td>
</tr>
<tr>
<td>RLA</td>
<td></td>
<td>XOR</td>
<td>Q8</td>
</tr>
<tr>
<td>RLC</td>
<td>Q8A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Definition of Symbols

The special symbols in the above table have the following significance:

R8 -- any of the set (A,B,C,D,E,H,L)
Q8A -- any of R8 or [(HL),(IX+D),(IY+D)]
Q8 -- any of Q8A or N8
D -- when used in (IX+D) or (IY+D), it must have a value between -128 and +127 decimal, inclusive
E -- used with jump relative instructions, it must have a value between -126 and +129 decimal, inclusive. It is usually an expression of the form 'SYMBOL-$'.
R16 -- any of the set (AF,BC,DE,HL,IX,IY,SP)
R16A -- any of the set (BC,DE,HL,SP)
R16B -- any of the set (BC,DE,IX,SP)
R16C -- any of the set (BC,DE,IY,SP)
R16D -- any of the set (BC,DE,HL,IX,SP)
R16E -- any of the set (AF,BC,DE,HL,IX,IY)
N3 -- a 3-bit integer; any of the set (0 .. 7)
N8 -- an 8-bit integer; any of the set (0 .. 255)
N16 -- a 16-bit integer; any of the set (0 .. 65535)
CF -- any of the set (Z,NZ,C,NC,PO,PE,M,P)
CFA -- any of the set (Z,NZ,C,NC)
Appendix B

Relocatable Object Format

This object file format is compatible with the MUMS object format implemented by Kominczak in the Universal Cross Assembler and the ARIES object format implemented by Conn in Project ARIES. This format additionally allows relocatable references by specifying an offset from the beginning of the current assembly and external references by specifying a name. All external symbols are listed in an External Symbol Dictionary (ESD) at the beginning of the object module to reduce lookahead problems. An object module can be described in SDL by the following:

```<OBJECT MODULE> : <ESD BLOCK>* <DATA BLOCK>* '$'
<ESD BLOCK> : '$S' <SYMBOL ENTRY>+ '&' <V8>
<SYMBOL ENTRY> : <RELATIVE DEFINITION> !
<ABSOLUTE DEFINITION> !
<REFERENCE DEFINITION>

<RELATIVE DEFINITION> : 'R' <NAME> <V16>
<ABSOLUTE DEFINITION> : 'A' <NAME> <V16>
<REFERENCE DEFINITION> : 'F' <NAME>
<NAME> : <ALPHA> <ALPHANUM>* <BLANK>
<ALPHA> : 'A' .. 'Z'
<ALPHANUM> : <ALPHA> ! '0' .. '9'
<BLANK> : "space character"
<V8> : <HEX DIGIT> <HEX DIGIT>
<V16> : <V8> <V8>
<HEX DIGIT> : 'A' .. 'F' ! '0' .. '9'
<data block> : 'S' <V16> <VALUE>* '&' <V8>
<VALUE> : <V8> !
<RELATIVE REFERENCE> !
<EXTERNAL REFERENCE>
<EXTERNAL REFERENCE>
<RELATIVE REFERENCE> : 'R' <V16>
```
All characters except '!', '&', and the checksum contribute to the modulo-256 checksum in the following manner:

<V8> -- the ASCII characters are converted to a single byte value, and the value of the byte is added to the checksum.

<ALPHA> -- the value is computed by adding the ordinal number ('A'=1) to 9. For example, for the character 'Z', 35 will be added to the checksum.

All other characters, including blank, are given the value zero.

All <V16>'s are not equal under the eyes of the linker. When in absolute load mode, the address immediately following the beginning hash character ('#') is given high-order byte first. All other word values are given low-order byte first.
Appendix C
Assembler Error Messages

A --- improper addressing mode usage in an expression

E --- null addressing mode encountered in an expression; this should never occur

F --- forward reference detected in operand of an EQU or DS

I --- invalid instruction mnemonic

M --- multiple definitions of a symbol have been encountered

N --- invalid character encountered in conversion of a numeric operand

O --- the all-inclusive operand error. This usually signifies improper usage of predefined register and flag names.

P --- missing 'END' statement

Q --- an equate cannot be made to an indirect symbol

R --- the range of a jump relative (-126 to +129 from the current instruction) has been exceeded

S --- a string has not been terminated

U --- an undefined symbol has been used in an operand field

Z --- an instruction has been used which is valid for the Z80 but not for the 8080. This error can occur only if the 'ONLY8080' pseudo-op has been used.
Appendix D

References

Conn, Richard, "ARIAN -- An Implementation of a Microcomputer Operating System", May 78; Master's Thesis, filed with the Graduate College, University of Illinois at Urbana-Champaign, Urbana, IL 61801. Also available through Defense Documentation Center, Alexandria, VA, under AD Number ADA060210.


INTEL Corporation, "INTEL 8080 Microcomputer Systems User's Manual", Sep 75; INTEL Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

MOSTEK Corporation, "MK 3880 Central Processing Unit", May 78; MOSTEK Corporation, 1215 W. Crosby Road, Carrollton, TX 75006.

MOSTEK Corporation, "Z80 Programming Manual", Dec 77; MOSTEK Corporation, 1215 W. Crosby Road, Carrollton, TX 75006.

ZILOG Corporation, "Z80-Assembly Language Programming Manual", Jan 78; ZILOG Corporation, 10460 Bubb Road, Cupertino, CA 95014.
CHAPTER 5

Source Code Listings

This chapter presents the source code listings of the Z80ASM system. Specifically, the programs addressed by this chapter are --

ZLDR    The Relocatable Linking Loader for the Z80 Assembler
Z80ASM  The Z80 Assembler
PROCEDURE ERROR (IDX : INTEGER);
BEGIN
  WRITE(LST,' ERROR - ');
  CASE IDX OF
  1 : WRITELN(LST,'IMPROPER ESD BLOCK IN PRIMARY FILE');
  2 : WRITELN(LST,'ESD CHECKSUM ERROR IN PRIMARY FILE');
  3 : WRITELN(LST,'ILLEGAL CHARACTER IN HEX CONVERT');
  4 : WRITELN(LST,'IMPROPER CHARACTER IN PRIMARY FILE DATA BLOCK');
  5 : WRITELN(LST,'DATA BLOCK CKSUM ERROR IN PRIMARY FILE');
  6 : WRITELN(LST,'IMPROPER TERMINATOR IN PRIMARY FILE');
  7 : WRITELN(LST,'MAXIMUM NUMBER OF EXTERNAL SYMBOLS \',\',MXSYM\',\',
               'HAS BEEN EXCEEDED');
  8 : WRITELN(LST,'EXTERNAL SYMBOL IN DATA BLOCK NOT DEFINED IN \',
               'SYMBOL BLOCK');
  9 : WRITELN(LST,'IMPROPER ESD ENTRY IN LIBRARY FILE');
 10: WRITELN(LST,'MAXIMUM NUMBER OF TEMPORARY SYMBOLS EXCEEDED');
  END;
  HALT;
END;

FUNCTION HEX (I : INTEGER) : CHAR;
BEGIN
  I := I MOD 16;
  IF I<0 THEN I := I + 16;
  IF (I IN \'0..9\') THEN HEX := CHR(I+ORD('0'))
    ELSE HEX := CHR(I-10+ORD('A'))
END;

FUNCTION VAL (CH : CHAR) : INTEGER;
BEGIN
  IF (CH IN \'0..9\') THEN VAL := ORD(CH) - ORD('0')
    ELSE IF (CH IN \'A..F\') THEN VAL := ORD(CH) + 10 - ORD('A')
    ELSE BEGIN
      ERROR(3);
      VAL := 0
    END
END;
(* PROCEDURE INIT DOES JUST THAT IT INITIALIZES. *)

PROCEDURE INIT:
VAR
CH : CHAR;
BEGIN
RESET(INPUT);
LDMODE := RELATIVE;
MAP := TRUE;
START := 0;
READ(CH):
IF (CH='R') THEN
BEGIN
LDMODE := RELATIVE;
WRITELN('RELATIVE LINK');
END
ELSE IF (CH='A') THEN
BEGIN
WRITE('ABSOLUTE LINK AT ');
LDMODE := ABSOLUTE;
START := 0;
READ(CH):
WHILE (CH IN '0'..'9', 'A'..'F') DO
BEGIN
WRITE(CH);
START := START+16 + VAL(CH);
READ(CH);
END;
END;
WRITELN;
PC := START;
LASTIN := 0;
NEW(STHEAD);
STHEAD-.NAME := '
STHEAD-.LSON := NIL;
STHEAD-.RSON := NIL;
STHEAD-.RESLVD := TRUE;
RESET(PRINTFILE);
REWRITE(LST);
REWRITE(TMPOBJ):
END;

(* FUNCTION UNRESOLVED RETURNS TRUE IF AN EXTERNAL REFERENCE *)
(* WHICH HAS NOT BEEN RESOLVED IS FOUND IN THE SYMBOL TABLE. *)

FUNCTION UNRESOLVED : BOOLEAN;
FUNCTION UNRES2 (ST : STEPTR) : BOOLEAN;
VAR
URFOUNDF : BOOLEAN; (* UNRESOLVED REFERENCE FOUND *)
BEGIN
WITH ST- DO BEGIN
(* CHECK PARENT NODE *)
URFOUNDF := NOT RESLVD;

(* IF PARENT IS RESOLVED, CHECK LEFT SON *)
IF (NOT UNFOUND) AND (LSON<>NIL) THEN UNRES2(LSON);
(* IF LEFT SON TREE IS RESOLVED, CHECK RIGHT SON *)
IF (NOT UNFOUND) AND (RSON<>NIL) THEN UNRES2(RSON);
UNRES2 := UNRES2
END
BEGIN
UNRESOLVED := UNRES2(STHEAD)
END;

******************************************************************************
FUNCTION SRCHST (SYM : ALFA; STEL : STEPTR) : BOOLEAN;
BEGIN
STFOUND := STEL; (* LEAVE POINTER TRAIL *)
WITH STEL DO
BEGIN
IF (SYM=NAME) THEN SRCHST := TRUE
ELSE IF (SYM<>NAME) THEN
IF (LSON<>NIL) THEN SRCHST := SRCHST(SYM,LSON)
ELSE SRCHST := FALSE
ELSE IF (RSYN<>NIL) THEN SRCHST := SRCHST(SYM,RSYN)
ELSE SRCHST := FALSE
END
END;

******************************************************************************
PROCEDURE INST INSERTS A SYMBOL INTO A SYMBOL TABLE.
(* THE SYMBOL IS VARIABLE TO SUPPORT TEMPORARY TABLES DURING *)
(* THE LIBRARY SEARCH. *)
******************************************************************************
PROCEDURE INST (SYM : ALFA; ADDR : INTEGER;
DEFINED : BOOLEAN; ST : STEPTR; MODE : ADRTYPE);
VAR PTR : STEPTR;
BEGIN
WITH ST- DO
BEGIN
IF SRCHST(SYM,ST) THEN
BEGIN
IF (DEFINED AND STFOUND-.RESLVD) THEN
WRITE(LST,' ERROR = MULTIPLE DEFINITIONS ENCOUNTERED FOR ',SYM)
ELSE IF (DEFINED THEN
BEGIN
STFOUND-.RESLVD := TRUE;
STFOUND-.ADM := MODE;
STFOUND-.VALUE := ADDR
END
END
ELSE BEGIN
   NEW(PTR);
   PTR-.NAME := SYM;
   PTR-.VALUE := ADDR;
   PTR-.RESVD := DEFINED;
   PTR-.ADM := MODE;
   PTR-.LSN := NIL;
   PTR-.RSON := NIL;
   IF (LASTSN=MXSYS) THEN ERROR(7)
   ELSE BEGIN
      LASTSN := LASTSN + 1;
      INS.LASTSN := PTR;
      PTR.ISN := LASTSN;
      END;
   IF (SYM<STFOUND-.NAME) THEN STFOUND-.LSN := PTR
   ELSE STFOUND-.RSON := PTR
   END

This code snippet is part of a larger program, possibly a compiler or a library that deals with symbol tables. The functions and procedures defined here are used for managing and manipulating symbol tables. The functions `NEW` and `WRITE` are used to add new entries to the table and write them to an object file, respectively. The symbol table is managed using a linked list, where each entry is a node that contains information about the symbol, such as its name, value, and dependencies.

The code includes several conditional statements that determine how the symbol table is updated and which entries are written to the object file. The `ERROR` function is called when an invalid operation is attempted, such as adding a symbol that already exists or trying to write to the end of the list when it is not empty.

The `OUTMAP` procedure is used to print a loader map, which is a representation of the symbol table in a format suitable for the loader. This map includes information about the symbol names, values, and their relationships within the program.
IF RSON<>NIL THEN OUTMAP2(RSON)
END
END;

BEGIN
MAPCOUNT := 0;
WRITELN(LST);
OUTMAP2(STHEAD);
WRITELN(LST);
END;

(* PROCEDURE LOPRIMFILE WILL LOAD THE PRIMARY FILE. IF ANY *)
(* EXTERNAL REFERENCES ARE NO IN THE SYMBOL TABLE, A FLAG IS *)
(* IS SET TO INDICATE A LIBRARY SEARCH IS NEEDED. *)

PROCEDURE LOPRIMFILE;
VAR
NAME : ALFA;
VALUE, RELPC, I : INTEGER;
CH,C1,C2,C3,C4 : CHAR;
LINEDONE, RELLOAD : BOOLEAN;

PROCEDURE READPRM (VAR CH : CHAR);
BEGIN
READ(PRIMFILE,CH);
IF (CH IN '0'..'9') THEN INCKSUM := INCKSUM + ORD(CH) - ORD('0')
ELSE IF (CH IN 'A'..'Z') THEN INCKSUM := INCKSUM + ORD(CH) - ORD('A') + 10
END;

PROCEDURE GETNAME;
VAR SYM : ARRAY '1'..111 OF CHAR;
I : INTEGER;
BEGIN
FOR I := 1 TO 10 DO SYM'I' := ' ';
I := 1;
READPRM(SYM'I');
I := I + 1
UNTIL SYM'I-111' = ' ';
PACK(SYM',NAME');
END;
/* FUNCTION GETVAL READS A WORD VALUE FROM THE PRIMARY FILE
* AND UPDATES THE CHECKSUM. */

FUNCTION GETVAL : INTEGER;
VAR
  I    : INTEGER;
BEGIN
  READ(PRMFILE,C1,C2,C3,C4);
  I := VAL(C1)*16 + VAL(C2) + VAL(C3) + 4096 + VAL(C4) + 256;
  INCKSUM := INCKSUM + I + I DIV 256;
  GETVAL := I;
END:

BEGIN
(* LOAD PRIMARY SYMBOL TABLE, IF ANY *)
INCKSUM := 0;
 Relbase := PC;
 Repeat READ(PRMFILE,CH)
   Until (CH='H') Or (CH='$');
While (PRMFILE='&') Do
  ReadPRM(CH); (* DISCARD 'S' *)
 LINEDONE := FALSE;
 Repeat
    ReadPRM(CH); (* GET INDICATOR *)
      If NOT (CH IN ['A','R','F','$']) Then ERROR(1)
 ELSE CASE CH OF
   'A' : BEGIN
      GETNAME;
      VALUE := GETVAL;
      INST(NAME,VALUE,TRUE,STHEAD,ABSOLUTE)
      END;
   'R' : BEGIN
      GETNAME;
      VALUE := GETVAL + Relbase;
      INST(NAME,VALUE,TRUE,STHEAD,LDMODE)
      END;
   'F' : BEGIN
      GETNAME;
      INST(NAME,0,FALSE,STHEAD,NULL)
      END;
   'S' : BEGIN
      READ(PRMFILE,CH,CH2);
      VALUE := VAL(CH)*16 + VAL(CH2);
      If VALUE <> (INCKSUM MOD 256) Then ERROR(2);
      LINEDONE := TRUE;
      INCKSUM := 0;
      Readln(PRMFILE);
      Repeat READ(PRMFILE,CH)
        Until (CH='H') Or (CH='$');
      END;
   END (* END CASE STATEMENT *)
   Until LINEDONE;
 END: (* END SYMBOL TABLE PROCESSING *)
WHILE (CH=HAS) DO
  BEGIN
  RELOAD := PRMFILE="R";
  IF RELOAD THEN BEGIN
    READPRM(CH);
    WRTMP(REL,D);
  END
  ELSE BEGIN
    READ(PRMFILE,C1,C2,C3,C4);
    RELPC := PC;
    PC := VAL(C1)*4096 + VAL(C2)*256 + VAL(C3)*16 + VAL(C4);
    INCKSUM := PC + PC DIV 256;
    WRTMP(ORG,PC);
  END;
  LINEDONE := FALSE;
  REPEAT
    READ(PRMFILE,CH);
    IF NOT(CH IN ['.0',.'9',.'A',.'F',.'R',.'X',.'&']) THEN ERROR(4)
    ELSE CASE CH OF
      '0',.'0',.'2',.'3',.'4',.'5',.'6',.'7',.'8',.'9',.'A',.'B',.'C',.'D',.'E',.'F':
        BEGIN
          READ(PRMFILE.CH2);
          I := VAL(CH1)*16 + VAL(CH2);
          INCKSUM := INCKSUM + I;
          WRTMP (BYTE, I);
          PC := PC + 1;
        END;
      'R': BEGIN
        INCKSUM := INCKSUM+ORD('0');
        I := GETVAL + RELBASE;
        WRTMP (AWORD,I);
        PC := PC + 2;
      END;
      'X': BEGIN
        INCKSUM := INCKSUM + 33;
        GETNAME;
        IF SRCST(NAME,STHEAD) THEN
          WITH STFOUND DO
            IF RESLV THEN
              IF (ADM=RELATIVE) THEN WRTMP(AWORD,VALUE)
              ELSE WRTMP (AWORD, VALUE)
            ELSE WRTMP (AWORD, ISN)
          ELSE ERROR(8);
        PC := PC + 2;
      END;
      'G': BEGIN
        READ(PRMFILE.CH,CH2);
        I := VAL(CH1)*16 + VAL(CH2);
        IF I>INCKSUM MOD 256 THEN ERROR(5);
        READ(PRMFILE);
        LINEDONE := TRUE;
        IF NOT RELOAD THEN PC := RELPC;
      REPEAT READ(PRMFILE.CH) UNTIL (CH=HAS) OR (CH='S');
      INCKSUM := 0;
OUTCHSUM := 0;
END;
END: (* END CASE STATEMENT *)
UNTIL LINEDONE: (* ONE LOAD BLOCK DONE *)
END: (* END DATA BLOCK PROCESSING *)
IF CH<>'S' THEN ERROR(6);
END;

***************************************************************************
(*) PROCEDURE RESOLVE SEARCHES THE LIBRARY FOR DEFINITIONS OF (*)
(*) THE REFERENCES FOUND IN THE PRIMARY FILE. IF A LIBRARY (*)
(*) ROUTINE PRODUCES UNRESOLVED REFERENCES, ANOTHER SEARCH (*)
(*) THROUGH THE LIBRARY WILL BE MADE. THE SEARCH IS ENDED WHEN (*)
(*) EITHER ALL REFERENCES ARE RESOLVED, OR ONE PASS IS MADE (*)
(*) THROUGH THE LIBRARY WITHOUT FINDING ANY NEEDED DEFINITIONS *)
***************************************************************************

PROCEDURE RESOLVE:
CONST
MXTMPS = 100;

VAR
TMSYM : ARRAY '1..MXTMPS' OF STE;
SYMCT : 0..MXTMPS;
NOFIND : BOOLEAN;
ONEPASS : BOOLEAN;
I : INTEGER;

NAME : ALFA;
CH,CH2,CT,C2,C3,C4 : CHAR;
LINEDONE : BOOLEAN;
VALUE : INTEGER;

***************************************************************************
(*) PROCEDURE INSTMP INSERTS A SYMBOL INTO THE TEMPORARY *)
(*) SYMBOL TABLE. *)
***************************************************************************

PROCEDURE INSTMP (S : ALFA; V : INTEGER; DEF : BOOLEAN; M : ADRTYPE);
BEGIN
IF SYMCT=MXTMPS THEN ERROR(10);
SYMCT := SYMCT + 1;
WITH TMSYM+SYMCT+1 DO
BEGIN
NAME := S;
VALUE := V;
RESOLVD := DEF;
ADM := M;
END;
END;

***************************************************************************
(*) PROCEDURE READOBJ READS ONE CHARACTER FROM THE LIBRARY *)
(*) FILE, AND UPDATES THE CHECKSUM ACCORDINGLY. *)
***************************************************************************
PROCEDURE READOBJ ( VAR CH : CHAR );
BEGIN
    READ(OBJLIB, CH);
    IF (CH IN '0'..'9') THEN INCKSUM := INCKSUM + (ORD(CH) - ORD('0'))
    ELSE IF (CH IN 'A'..'Z') THEN INCKSUM := INCKSUM + (ORD(CH) - ORD('A')) + 10
END;

************************************************************
(* PROCEDURE GETNAME READS AN ESD NAME ENTRY AND PACKS IT *)
(* INTO VARIABLE 'NAME'. *)
************************************************************

PROCEDURE GETNAME;
VAR
    SYM : ARRAY[1..11] OF CHAR;
    I : INTEGER;
BEGIN
    FOR I := 1 TO 10 DO SYM[I] := ' ';
    I := 1;
    REPEAT
        READOBJ(SYM[I]);
        I := I + 1
    UNTIL SYM[I] = ' ';
    PACK(SYM, I, NAME);
END;

************************************************************
(* FUNCTION GETVAL READS A WORD VALUE FROM THE LIBRARY FILE *)
(* AND UPDATES THE CHECKSUM. *)
************************************************************

FUNCTION GETVAL : INTEGER;
VAR
    I : INTEGER;
BEGIN
    READ(OBJLIB, C1, C2, C3, C4);
    I := VAL(C1) + 16 + VAL(C2) + VAL(C3) + 4096 + VAL(C4) + 256;
    INCKSUM := INCKSUM + I + I DIV 256;
    GETVAL := I;
END;

************************************************************
(* PROCEDURE LOADSYMS READS THE ESD FROM THE NEXT LIBRARY *)
(* PARTITION. *)
************************************************************

PROCEDURE LOADSYMS;
BEGIN
    SYMCNT := 0;
    INCKSUM := 0;
    RELBASE := PC;
    REPEAT
        READ(OBJLIB, CH)
        UNTIL (CH='S') OR (CH='$')
    WHILE (OBJLIB='S') DO
    BEGIN
        READOBJ(CH); (* DISCARD 'S' *)
    END
END;
LINEDONE := FALSE;
REPEAT
READOBJ(CH): (* GET INDICATOR *)
   IF NOT (CH IN ('A','R','F','&')) THEN ERROR(9)
ELSE CASE CH OF
   'A': BEGIN
      GETNAME;
      VALUE := GETVAL;
      INSTMP(NAME,VALUE,TRUE,ABSOLUTE)
   END;
   'R': BEGIN
      GETNAME;
      VALUE := GETVAL + RELBASE;
      INSTMP(NAME,VALUE,TRUE,LOMODE)
   END;
   'F': BEGIN
      GETNAME;
      INSTMP(NAME,0,FALSE,NUL)
   END;
   'A': BEGIN
      READ(OBJLIB,CH,CH2);
      VALUE := VAL(CH)*16 + VAL(CH2);
      IF VALUE <> (INCKSUM MOD 256) THEN ERROR(2);
      LINEDONE := TRUE;
      INCKSUM := 0;
      READLN(OBJLIB);
      REPEAT READ(OBJLIB,CH)
         UNTIL (CH='H') OR (CH='I');
   END;
   END (* END CASE STATEMENT *)
   UNTIL LINEDONE;
END (* END SYMBOL TABLE PROCESSING *)
END;

FUNCTION URINLIST: BOOLEAN;
VAR
   I : INTEGER;
   NOMATCH : BOOLEAN;
BEGIN
   NOMATCH := TRUE;
   I := 0;
   WHILE (NOMATCH) AND (I<SYMCT) DO
      BEGIN
         I := I + 1;
         IF SRCNST(TMPSYM:I1,NAMC,STHEAD) THEN
            IF NOT (STFIND-.RESLVD) THEN NOMATCH := FALSE;
   END;
   URINLIST := NOT NOMATCH;
END:
PROCEDURE LOADDATA:
VAR
  RELPC,I       : INTEGER;
  LINEDONE,RELOAD : BOOLEAN;
BEGIN
  WHILE (CH=HAS) DO
  BEGIN
    RELOAD := OBJLIB='R';
    IF RELOAD THEN BEGIN
      READOBJ(CH);
      WRTMP(REL,0);
    END
    ELSE BEGIN
      READ(OBJLIB,C1,C2,C3,C4);
      RELPC := PC;
      PC := VAL(C1)*4096 + VAL(C2)*256 + VAL(C3)*16 + VAL(C4);
      INCKSUM := PC + PC DIV 256;
      WRTMP(ORG,PC);
      END;
    LINEDONE := FALSE;
  REPEAT
    READ(OBJLIB,CH);
    IF NOT(CH IN '0'..'9','A'..'F','R','X','&'1) THEN ERROR(4)
    ELSE CASE CH OF
      '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F': BEGIN
        READ(OBJLIB,CH2);
        I := VAL(CH)+16 + VAL(CH2);
        INCKSUM := INCKSUM + I;
        WRTMP (BYTE,1);
        PC := PC + 1;
      END;
      'R': BEGIN
        INCKSUM := INCKSUM+ORD('0');
        I := GETVAL + RELBASE;
        WRTMP (RWORD,1);
        PC := PC + 2;
      END;
      'X': BEGIN
        INCKSUM := INCKSUM + 33;
        GETNAME;
        IF SRCST(NAM,STHEAD) THEN
          WITH SFOUND- DO
          IF RESLOD THEN
            IF (ADM=RELATIVE) THEN WRTMP(RWORD,VALUE)
            ELSE WRTMP (AWORD,VALUE)
            ELSE WRTMP (XWORD,ISN)
            ELSE ERROR(8);
        END
        PC := PC + 2;
      END;
    END;
  END;
END.
END;
'&': BEGIN
READ(OBJLIB,CH,CH2);
I := VAL(CH)*16 + VAL(CH2);
IF I<>INCKSUM MOD 256 THEN ERROR(5);
READLN(OBJLIB);
LINEDONE := TRUE;
IF NOT RELLOAD THEN PC := RELPC;
REPEAT READ(OBJLIB,CH) UNTIL (CH=HAS) OR (CH='$');
INCKSUM := 0;
OUTCKSUM := 0;
END;
END; (* END CASE STATEMENT *)
UNTIL LINEDONE; (* ONE LOAD BLOCK DONE *)
END; (* END DATA BLOCK PROCESSING *)
IF CH>'$' THEN ERROR(6)
ELSE GETSEG(OBJLIB);
END;

BEGIN
RESET(OBJLIB);
ONEPASS := FALSE;
NOFIND := TRUE;
WHILE (NOT ONEPASS) AND UNRESOLVED DO
  IF EOF(OBJLIB) THEN
    BEGIN
      ONEPASS := NOFIND;
      NOFIND := TRUE;
      RESET(TMPOBJ);
      END
      ELSE BEGIN
          LOADSYMS;
          IF URMINLIST THEN
              BEGIN
                FOR I := 1 TO SYMCT DO
                WITH TMPSYM*II DO INSST(NAME,VALUE,RESLVD,STHEAD,ADM);
                NOFIND := FALSE;
                LOADDATA;
              END
              ELSE GETSEG(OBJLIB);
              END
END;

(*----------------------------------------------------------------*)
(*  PROCEDURE FLUSH OUTPUTS THE CHECKSUM INTO THE OBJECT FILE*)
(*----------------------------------------------------------------*)

PROCEDURE FLUSH:
BEGIN
  WRITELN(OBJ,'&',HEX(OUTCKSUM DIV 16),HEX(OUTCKSUM));
  OBJCNTR := 0;
  OUTCKSUM := 0;
END;

(*----------------------------------------------------------------*)
(* PROCEDURE EMIT PUTS A SINGLE BYTE INTO THE OUTPUT OBJECT *)
(* FILE. *)

PROCEDURE EMIT(V : INTEGER);
BEGIN
  IF OBJCNT>OBJMAX THEN FLUSH;
  IF OBJCNT = 0 THEN
    BEGIN
      IF LDMODE=RELATIVE THEN
        BEGIN
          WRITE(OBJ,HAS,'R');
          OUTCKSUM := ORD('0');
          OBJCNT := 2;
        END;
    ELSE
      BEGIN
        WRITE(OBJ,HAS,HEX(PC DIV 4096),HEX(PC DIV 256),HEX(PC DIV 16),
          HEX(PC));
        OUTCKSUM := PC + PC DIV 256;
        OBJCNT := 5;
      END;
    END;
  END;
  END;

PROCEDURE EMITW(V : INTEGER; M : ADRTYPE);
BEGIN
  IF (M=RELATIVE) THEN
    BEGIN
      IF OBJCNT>OBJMAX-5 THEN FLUSH;
      IF OBJCNT=0 THEN
        BEGIN
          WRITE(OBJ,'AS','R');
          OUTCKSUM := ORD('0');
          OBJCNT := 2;
        END;
      END;
      WRITE(OBJ,'R');
      OUTCKSUM := OUTCKSUM + ORD('0');
      OBJCNT := OBJCNT + 1;
    END;
  END;
END;

(* PROCEDURE PASSWDO WILL COPY THE TEMPORARY OBJECT FILE TO *)
(* THE OUTPUT OBJECT FILE. COMPLETING RESOLUTION OF ALL *)
PROCEDURE PAStTwo:
VAR CH : CHAR;
X : OBJEL
* RELPC : INTEGER;
BEGIN
RESET(TMPOBJ);
REWIT(OBJ);
OUTMODE := RELATIVE;
PC := START;
RELPC := START;
WHILE NOT EOF(TMPOBJ) DO
BEGIN
READ(TMPOBJ,X);
CASE X.TYPE OF
  BYTE : EMIT(X.VAL);
  ANORD: EMITW(X.VAL,ABSOLUTE);
  RWORD: EMITW(X.VAL,LDMODE);
  WWORD: WITH ISNS X.VAL DO
    IF ADM=RELATIVE THEN EMITW(VALUE,LMODE)
    ELSE EMITW(VALUE,ABSOLUTE);
  REL : BEGIN
    IF (OUTMODE=ABSOLUTE) THEN PC := RELPC;
    IF (OBJCNT<>0) AND (OUTMODE=ABSOLUTE) THEN FLUSH;
    OUTMODE := RELATIVE;
    END;
  ORG : BEGIN
    PC := X.VAL;
    OUTMODE := ABSOLUTE;
    IF OBJCNT<>0 THEN FLUSH;
    END;
  END;
  END;
  IF OBJCNT<>0 THEN FLUSH;
  WRITEN(OBJ,'$');
END:

BEGIN
INIT; HAS := CHR(48); (* INITIALIZE AND DEFINE HAS CHAR (HASH) *)
LDPRIMFILE;
IF UNRESOLVED THEN RESOLVE;
OBJCNT := 0;
IF NOT UNRESOLVED THEN PAStTwo;
IF MAP THEN OUTMAP;
END.
END.
Source Code Listings

Z80ASM -- The Z80 Assembler
VERSION 1.1
WRITTEN BY GEORGE LEHMANN
UNIVERSITY OF ILLINOIS
JANUARY, 1978

VERSION 1.2 -- MINOR MODIFICATIONS MADE TO VERSION 1.1
MODIFIED AND MAINTAINED BY RICHARD CONN
USA SATELLITE COMMUNICATIONS AGENCY
MAY, 1979

PROGRAM OPERATION --
THE BASIC PROGRAM LOOP IS CONTAINED IN PROC PASS. AFTER
GETTING A LINE OF TEXT FROM INLINE (READS INPUT) OR MACLINE
(EXPANDS MACROS), THE START-OF-LINE SYMBOL (IF THERE IS ONE)
AND THE INSTRUCTION MNEMONIC ARE PICKED OFF. IF THE INSTRUCTION
IS NOT IN THE OPCODE TABLE, GETMAC IS CALLED TO START EXPANSION
OF THE MACRO (OR INDICATE AN ERROR IF THE MACRO CAN'T BE FOUND)
IF THE INSTRUCTION IS IN THE TABLE, THE NOPHOS VALUE IS USED
TO EVALUATE THE CORRECT NUMBER OF OPERANDS, AND THE SWITCH
VALUE IS USED TO JUMP INTO THE EVALUATION CASE STATEMENT.
THERE IS A SEPARATE CASE FOR EACH INSTRUCTION MNEMONIC.
HERE, OPERAND TYPES ARE CHECKED FOR LEGAL COMBINATIONS AND THE
APPROPRIATE OBJECT CODE IS GENERATED USING VARIOUS EMIT
ROUTINES.
FIVE ADDRESS MODES ARE USED FOR SYMBOLIC VALUES, RELATIVE
IS FOR WHEN RELOCATABLE CODE (OR DATA) IS BEING GENERATED.
THE VALUE SHOWN IN THE ASSEMBLY LISTING FOR SUCH A LOCATION
IS RELATIVE TO THE BEGINNING OF THE RELOCATABLE ASSEMBLY.
ABSOLUTE IS FOR WHEN CODE (OR DATA) IS PLACED INTO AN ABSOLUTE
LOCATION BY AN ORG INSTRUCTION. THE VALUE SHOWN WITH THIS
IS THE MEMORY ADDRESS USED BY ALL REFERENCES TO THE LOCATION.
EXTERNAL MODES ARE INDICATED WHEN AND EXIT INSTRUCTION IS USED
TO DEFINE A SYMBOL. IF THE SYMBOL IS ALSO DEFINED ELSEWHERE
IN THE SAME ASSEMBLY, IT BECOMES AND EXTDEFX OR EXTDEFA,
DEPENDING ON THE ADDRESS OF THE SYMBOL BEFORE BEING MADE
EXTERNAL. IF THE SYMBOL IS NOT DEFINED ELSEWHERE, IT BECOMES
AN EXTF, ALLOWING RESOLUTION OF ITS ADDRESS TO BE DEFERRED
UNTIL LOAD TIME.
PROGRAM Z80ASM (SOURCE,LST,OBJ,OUTPUT);

CONST
QWIDTH = 110;
IWIDTH = 60;
OFFSET = 30;
HEXMAX = 25;
OBJMAX = 94;
SMAX = 10; (*. MAXIMUM SYMBOL LENGTH *);
LMAX = 56; (*. LISTING PAGE SIZE *);
MSTACK = 50;
HSkip = 2;

TYPE
WORD = 0..65535;
SETI = SET OF 1..40;
BYTE = 0..255;
STPTR = STE;
OTPTR = OTE;
SWTYPE = 1..40;
OPNUM = 0..2;
ADRTYPE = (NULL,RELATIVE,ABSOLUTE,EXTDEFA,EXTDEFR,EXTREF,REGISTER,IMMEDIATE);
REFLINK = -REF;
REF = RECORD
  ADDR : WORD;
  NEXT : REFLINK
END:
STE = PACKED RECORD
  (* SYMBOL *)
  NAME : ALFA;
  VALUE :WORD;
  PDEF,P2,SPECIAL : BOOLEAN;
  ADMODE : ADRTYPE;
  REFS : REFLINK;
  LSON,RSN : OTPTR
END:
OTE = RECORD
  (* OPCODE *);
  NAME : ALFA;
  DFLT : BYTE;
  SWITCH : SWTYPE;
  NOPNDOS : OPNUM;
  LSON,RSN : OTPTR
END:
OPNDRYP = (R8,RA,RAF,RBD,RH,RSP,RX,RY,IC,IBD,ILH,ISP,
           IX,IY,CF,CFI,CF2,RIR,CON,ICON,RPC);
OPSET = SET OF OPNDRYP;
OPERS = (NOOP,PLUS,MINUS,ATIMES,ADIV,AMOD,ANDL,O RL);

VAR
PASS1,PASS2 : BOOLEAN;
STFOUND,STHEAD : STPTR;
OTPNDOS,OTPHEAD : OTPTR;
OBJ,SOURCE,LST : TEXT;
LINECNT,PC,IPPOS : INTEGER;
HEXLOC.OBJCNT : INTEGER;
STATIME.PGCNT : INTEGER;
EPOS.ERRCNT : INTEGER;
RELSAVE.CKSUM : INTEGER;
SPOS.PFOUND : INTEGER;
PSYM.PINSTR : ALFA;
VTIME.VDATE : ALFA;
PEXT : ALFA;
STOP.REL.INMACRO : BOOLEAN;
NOTEND.NOTBLNK : BOOLEAN;
OPSET1.OPSET2 : OPSET;
VAL1.VAL2 : NORD;
DEFLT.DBYTE : INTEGER;
ASNTYP : ADRTYPE;
NOXREF.LIST : BOOLEAN;
PRGEM.SYMTAB : BOOLEAN;
P2XREF.NOSUPRES : BOOLEAN;
PDVAL : ARRAY[1..311] OF INTEGER;
POSET : ARRAY[1..311] OF OPSET;
OLINE : ARRAY[1..WIDTH] OF CHAR;
TITLE.HEADER : ARRAY[1..61] OF ALFA;
EFLG : ARRAY[1..13] OF CHAR;
ASC : ARRAY[0..63] OF BYTE;
INP.STR : ARRAY[1..WIDTH] OF CHAR;
SYMBL.INSTR : ARRAY[1..MAXI] OF CHAR;
PRIORITY : ARRAY[OPERS] OF INTEGER;
ONLY8800 : BOOLEAN;
HASH,QUOTE,COLON : CHAR;

*******************************************************************************
(* INTERNAL DEFINITIONS FOR LOGICAL OPERATION PROCEDURES *)
*******************************************************************************

PROCEDURE LAND(A,B:INTEGER; VAR C:INTEGER);
BEGIN
  C := A*B;
END;

PROCEDURE LOR(A,B:INTEGER; VAR C:INTEGER);
BEGIN
  C := A+B;
END;

*******************************************************************************
(* THIS PROCEDURE INCREMENTS THE ERROR COUNT AND INDICATES *)
(* IT IN THE LISTING. *)
*******************************************************************************

PROCEDURE ERROR(INDICATOR : INTEGER);
BEGIN
  OLINE'31 := HASH;
  ERRCNT := ERRCNT + 1;
  IF EPOS<OFFSET THEN OLINE'EPOS := EFLG'INDICATOR!!
  EPOS := EPOS + 1
FUNCTION HEX (I : INTEGER) : CHAR;
BEGIN
  1 := I MOD 16;
  IF I<0 THEN 1 := I + 16;
  IF I<10 THEN HEX := CHR(ORD('0')+1)
  ELSE HEX := CHR(ORD('A')+I-10)
END;

FUNCTION SRCHST(SYM : ALFA; PTR : STPTR) : BOOLEAN;
BEGIN
  STFOUND := PTR; (* POINTER TRAIL*)
  IF SYM = PTR-.NAME THEN SRCHST := TRUE
  ELSE IF SYM < PTR-.NAME THEN
    IF PTR-.LSON = NIL THEN SRCHST := FALSE
    ELSE SRCHST := SRCHST(SYM, PTR-.LSON)
  ELSE IF PTR-.RSON = NIL THEN SRCHST := FALSE
  ELSE SRCHST := SRCHST(SYM, PTR-.RSON)
END;

PROCEDURE INSST INSERTS ALL SYMBOLS INTO THE SYMBOL TABLE;
*  NO ERRORS ARE INDICATED DURING PASS 1 IF MULTIPLE *
*  ATTEMPTS ARE MADE TO DEFINE A SYMBOL. DURING PASS 2, THE *
*  FIRST INSERTION OF A SYMBOL WILL SIMPLY RESULT IN TURNING *
* ON THE P2 FLAG. ANY SUBSEQUENT INSERTION ATTEMPT WILL *
* CAUSE A MULTIPLE-DEFINITION ERROR TO BE IndICATED. *
* IF A SYMBOL HAS ALREADY BEEN DEFINED BY AN EXT *
* INSTRUCTION, THE ADDRESS MODE OF THE SYMBOL WILL BE *
* CHANGED TO EXTDEFR OR EXTDEFA, DEPENDING ON THE INPUT *
* ADDRESS MODE, ADM. *

PROCEDURE INSST(SYM : ALFA; VAL : INTEGER; PREDEF : BOOLEAN; 
ADM : ADRTYPE; SPCL : BOOLEAN);
VAR NEWSYM : STPTR;

BEGIN
  IF SrchSt(SYM, STHEAD) THEN
    WITH STFOUND DO BEGIN
      IF PASS2 AND P2 THEN ERROR(1):
      IF PASS1 AND (ADMODE=EXTREF) THEN
        BEGIN
          IF ADM=RELATIVE THEN ADMODE := EXTDEFR
          ELSE ADMODE := EXTDEFA;
          VALUE := VAL;
        END
      P3 := (PASS2 AND (ADM<>EXTREF)) OR P2
    END ELSE BEGIN
      NEW(NEWSYM);
      WITH NEWSYM DO BEGIN
        NAME := SYM;
        VALUE := VAL;
        P2 := FALSE;
        PDEF := PDEF;
        SPECIAL := SPCLI;
        LSON := NIL;
        RSON := NIL;
        REFS := NIL;
        ADMODE := ADM;
      END;
      IF SYM<>STFOUND-.NAME THEN STFOUND-.LSON := NEWSYM
      ELSE STFOUND-.RSON := NEWSYM
    END;
  END;
END; (* END PROCEDURE INST *)

(* PROCEDURE ZPAGE SIMPLY PRINTS THE HEADER WHEN REQUESTED. *)
(* ********************************************** *)

PROCEDURE ZPAGE;

VAR I : INTEGER;

BEGIN
  Writeln(LST);
  IF PGCNT<>1 THEN Writeln(LST,'    PROGRAM ERRORS = ',ERRCNT:5);
  Writeln(LST);
  Writeln(LST,'1');
  Writeln(LST,'2');
  LINCNT := 1;
  FOR I := 1 TO HSKIP DO BEGIN
    Writeln(LST);
    LINCNT := LINCNT + 1
  END;
  Writeln(LST,'Z-BO ASSEMBLER V1.2 ');
  FOR I := 1 TO 6 DO Writeln(LST,TITLE:10);
  Writeln(LST,' PAGE NUMBER ',PGCNT:5);
PGCNT := PGCNT + 1;
WRITE(LST, ',
FOR I:=1 TO 6 DO WRITE(LST, HEADER[I]);
WRITELN(LST);
WRITELN(LST);
LINECNT := LINECNT + 5
END;

******************************************************************************
(*) PROCEDURE DUMPST RECURSIVELY Dumps SYMBOL Table Entries (*)
(*) ACCORDING TO THE FLAG, NOXREF, IF NOXREF IS TRUE, A (*)
(*) COMPACTED Symbol Table Is Printed, Giving Only The Value (*)
(*) OF EACH Symbol. IF NOXREF IS FALSE, A LARGER Listing Is (*)
(*) PRODUCED, Giving Mode, VALUE, AND ALL References FOR EACH*)
(*) Symbol. Since InOrder Traversal Is Used, The Symbol (*)
(*) Will Naturally Be Sorted.
******************************************************************************

PROCEDURE DUMPST( PTR : STPTR );

VAR
SLINK : REFLINK;

PROCEDURE WHEX(VAL : WORD):
BEGIN
WRITE(LST, ',',HEX(VAL DIV 4096),HEX(VAL DIV 256),
HEX(VAL DIV 16), HEX(VAL))
END;

PROCEDURE WRTYPE(TYP : AORTYPE):
BEGIN
CASE TYP OF
RELATIVE: WRITE(LST, ' RELATIVE ');
ABSOLUTE: WRITE(LST, ' ABSOLUTE ');
EXTDEF A : WRITE(LST, ' EXT DEF A ');
EXTDEFR : WRITE(LST, ' EXT DEF R ');
EXTREF : WRITE(LST, ' EXT REF ');
REGISTER : WRITE(LST, ' REGISTER ');
IMMEDIATE: WRITE(LST, ' ABSOLUTE ')
END

PROCEDURE SYMPAGE;
BEGIN
WRITELN(LST);
ZPAGE;
IF NOXREF THEN
WRITELN(LST, ' SYMBOL Table')
ELSE WRITELN(LST, ' SYMBOL VALUE ADDR TYPE REFERENCES');
WRITELN(LST);WRITELN(LST);
LINECNT := LINECNT + 5;
SPOS := 0
END;

BEGIN

*/
WITH PTR DO BEGIN
  IF LS0N = NIL THEN DUMPST(LSON);
  IF LINECNT = LMAX THEN SYMPAGE;
  IF NOT SPECIAL THEN
    IF NSXREF THEN BEGIN
      IF SPOS = 5 THEN BEGIN
        IF LINECNT = LMAX THEN SYMPAGE;
        WRITELN(LST);
        LINECNT := LINECNT + 1;
        SPOS := 0
      END;
      WRITE(LST, ', NAME);
      WHX (VALUE);
      SPOS := SPOS + 1
    END;
    ELSE BEGIN
      SPOS := 0;
      WRITE(LST, ', NAME);
      WHX (VALUE);
      WR_TYPE(ADMODE);
      SLINK := REFS;
      WHILE SLINK = NIL DO
        BEGIN
          IF SPOS = 15 THEN BEGIN
            IF LINECNT = LMAX THEN SYMPAGE;
            WRITELN(LST);
            WRITE(LST, ',
          END;
          SPOS := SPOS + 1;
          WHX (SLINK, ADDR);
          SLINK := SLINK, NEXT
        END;
        LINECNT := LINECNT + 1;
        WRITELN(LST)
      END;
      IF RS0N = NIL THEN DUMPST(RSON)
    END;
END;

*******************************************************************************
(* PROCEDURE PDSF IS MERELY A CONVENIENCE FOR INSERTING *)
(* PREDEFINED ENTRIES INTO THE SYMBOL TABLE. SINCE ALL *)
(* PREDEFINED ENTRIES USE THEIR VALUE TO REFERENCE ARRAYS *)
(* POSET AND PDVAL, THIS ROUTINE ALSO BUILDS THESE ARRAYS. *)
(* THESE EXTRA ARRAYS ALLOW THE SYMBOL TABLE ENTRY TO BE *)
(* SMALLER FOR NON-PREDEFINED SYMBOLS, WITHOUT USING VARIANT *)
(* RECORDS. THESE ARRAYS ALSO CONTAIN EXTRA DEFINITIONS *)
(* (VERY ORDER DEPENDENT) FOR THE INDIRECT REFERENCES USED *)
(* IN MANY INSTRUCTIONS. SEE PROC GETOP FOR MORE DETAIL. *)
*******************************************************************************

PROCEDURE PDSF(SYM : ALFA; IDX,VAL : INTEGER;
OP : OPSET);

BEGIN
  INST(SYM,IDX,TRUE,ABSOLUTE,TRUE);
  PDVAL,IDX := VAL;
  PDSET,IDX := OP
END:

(* PROCEDURE SRCHOT FUNCTIONS MUCH THE SAME FOR THE OPCODE *)
(* TABLE AS SRCHST DOES FOR THE SYMBOL TABLE. *)

FUNCTION SRCHOT(SYM : ALFA; PTR : OTPTR) : BOOLEAN;

BEGIN
  OTFOUND := PTR; (* POINTER TRAIL *)
  IF SYM = PTR-.NAME THEN SRCHOT := TRUE
  ELSE IF SYM < PTR-.NAME THEN
    IF PTR-.LSO = NIL THEN SRCHOT := FALSE
    ELSE SRCHOT := SRCHOT(SYM, PTR-.LSO)
    ELSE IF PTR-.RSON = NIL THEN SRCHOT := FALSE
    ELSE SRCHOT := SRCHOT(SYM, PTR-.RSON)
END:

(* PROCEDURE INSOT INSERTS OPCODES INTO THE BINARY-TREE *)
(* OPCODE TABLE. IF A DUPLICITY OCCURS, IT IS A MAJOR ERROR *)
(* AND THE PROGRAM IS HALTED. *)

PROCEDURE INSOT(SYM : ALFA; DEFLT : BYTE;
                 ISWCH : SWTYPE; NOPNS : OPNUM);

VAR NEWSYM : OTPTR;

BEGIN
  IF SRCHOT(SYM,OTHEAD) THEN HALT:
  TIME(VTIME);
  DATE(VDATE);
  NEW(NEWSYM):
  WITH NEWSYM- DO BEGIN
    NAME := SYM;
    DEFLT := DEFLT;
    SWITCH := ISWCH;
    NOPNS := NOPNS;
    LSON := NIL;
    RSON := NIL
    END:
  IF SYM < OTFOUND-.NAME THEN OTFOUND-.LSO := NEWSYM
  ELSE OTFOUND-.RSON := NEWSYM
END:

(* PROCEDURE INIT IS RESPONSIBLE FOR: *)
PROCEDURE INIT;

VAR I : INTEGER;

BEGIN
  HASH := CHR(48); (* HASH CHAR *).
  COLON := CHR(0); (* COLON CHAR *).
  SQUOTE := CHR(56); (* SINGLE QUOTE CHAR *).
  ONLY8080 := FALSE;
  NEW(STHEAD);
  WITH STHEAD DO BEGIN
    NAME := 'I';
    VALUE := 30;
    PDEF := TRUE;
    SPECIAL := TRUE;
    P2 := FALSE;
    LSON := NIL;
    RSON := NIL;
    REFS := NIL;
  END;
  PDVAL'301 := 0;
  PDSET'301 := 'RIR';
  PDSTATE'C' := 8, 1, 'RB,CF,CF11';
  PDSTATE'NC' := 24, 2, 'CF,CF11';
  PDSTATE'B' := 2, 0, 'RBI';
  PDSTATE'E' := 5, 3, 'RBI';
  PDSTATE'A' := 1, 7, 'RA,RBI';
  PDSTATE'AF' := 7, 3, 'RAFI';
  PDSTATE'BC' := 9, 0, 'RBDI';
  PDSTATE'D' := 4, 2, 'RBI';
  PDSTATE'DE' := 10, 1, 'RBDI';
  PDSTATE'M' := 6, 4, 'RBI';
  PDSTATE'HL' := 11, 2, 'RHI';
  PDSTATE'LY' := 14, 2, 'RYI';
  PDSTATE'IX' := 13, 2, 'RXI';
  PDSTATE'LL' := 3, 5, 'RBI';
  PDSTATE'M' := 29, 7, 'CFI';
  PDSTATE'P' := 28, 6, 'CFI';
  PDSTATE'NZ' := 22, 0, 'CF,CF11';
  PDSTATE'P0' := 26, 4, 'CFI';
  PDSTATE'PE' := 27, 5, 'CFI';
  PDSTATE'CP' := 12, 3, 'RSP1';
  PDSTATE'R' := 31, 8, 'RIR';
  PDSTATE'Z' := 23, 1, 'CF,CF11';
  PDVAL'151 := 0;
  PDSET'151 := 'IC1';
  PDVAL'161 := 0;
  PDSET'161 := 'IBD';
  PDVAL'171 := 1;
  PDSET'171 := 'IBD';
  PDVAL'181 := 0;
  PDSET'181 := 'IH,RBI';
NEW(OHEAD);
WITH OHEAD DO BEGIN
  NAME := 'LD
  DFLT := 0;
  SWITCH := 23;
  NOPNDS := 2;
  LSON := NIL;
  RSON := NIL;
END;

INSOT('CALL', 196, 7, 1); INSOT('ADD', 128, 4, 2);
INSOT('CCF', 63, 1, 0); INSOT('AND', 47, 1, 0);
INSOT('DAA', 39, 1, 0); INSOT('DI', 243, 1, 0);
INSOT('EI', 251, 1, 0); INSOT('EXX', 217, 1, 0);
INSOT('HALT', 118, 1, 0); INSOT('NOP', 0, 1, 0);
INSOT('RLA', 23, 1, 0); INSOT('RLCA', 7, 1, 0);
INSOT('RR', 31, 1, 0); INSOT('RRCA', 15, 1, 0);
INSOT('SCF', 55, 1, 0); INSOT('CPO', 169, 2, 0);
INSOT('CPR', 185, 2, 0); INSOT('CPI', 161, 2, 0);
INSOT('CPR', 177, 2, 0); INSOT('INO', 170, 2, 0);
INSOT('IND', 186, 2, 0); INSOT('INI', 152, 2, 0);
INSOT('IN', 176, 2, 0); INSOT('LD', 168, 2, 0);
INSOT('LDD', 184, 2, 0); INSOT('LDI', 160, 2, 0);
INSOT('LD', 176, 2, 0); INSOT('NEG', 68, 2, 0);
INSOT('OUTD', 171, 2, 0); INSOT('OUTR', 139, 2, 0);
INSOT('OUTI', 163, 2, 0); INSOT('OTR', 179, 2, 0);
INSOT('RET', 77, 2, 0); INSOT('RETN', 69, 2, 0);
INSOT('RLD', 111, 2, 0); INSOT('RRD', 103, 2, 0);
INSOT('ADC', 136, 3, 2); INSOT('AND', 160, 5, 1);
INSOT('BIT', 64, 6, 2); INSOT('OR', 176, 5, 1);
INSOT('RES', 128, 6, 2); INSOT('SET', 192, 6, 2);
INSOT('SRR', 186, 5, 1); INSOT('RET', 192, 8, 0);
INSOT('CP', 184, 5, 1); INSOT('UP', 194, 9, 1);
INSOT('JR', 32, 10, 1); INSOT('DEC', 5, 11, 1);
INSOT('INC', 4, 12, 1); INSOT('EX', 0, 13, 2);
INSOT('IN', 70, 14, 1); INSOT('OUT', 65, 15, 2);
INSOT('POP', 193, 16, 1); INSOT('PUSH', 197, 16, 1);
INSOT('RL', 16, 17, 1); INSOT('RLC', 0, 17, 1);
INSOT('RR', 24, 17, 1); INSOT('RRC', 8, 17, 1);
INSOT('SBC', 152, 16, 2); INSOT('SLA', 32, 17, 1);
INSOT('SRA', 40, 17, 1); INSOT('SRL', 86, 17, 1);
INSOT('SUB', 144, 5, 1); INSOT('DJNZ', 16, 20, 1);
INSOT('IN', 64, 21, 2); INSOT('RST', 199, 22, 1);
INSOT('EQU', 0, 24, 1); INSOT('ORG', 0, 25, 1);
INSOT('REL', 0, 26, 0); INSOT('END', 0, 27, 0);
INSOT('DB', 0, 26, 0); INSOT('DW', 0, 29, 0);
INSOT('DS', 0, 30, 1); INSOT('EXT', 0, 31, 0);
INSOT('TITLE', 0, 32, 0); INSOT('MACRO', 0, 33, 0);
INSOT('MEND', 0, 34, 0); INSOT('GEN', 0, 39, 0);
INSOT('NOGEN', 0, 36, 0); INSOT('LIST', 0, 37, 0);
INSOT('NOLIST', 0, 38, 0); INSOT('XREF', 0, 39, 0);
INSOT('NOSYM', 0, 19, 0); INSOT('PAGE', 0, 40, 0);
PROCEDURE MACLINE;
BEGIN
HALT
END;

PROCEDURE GETMAC;
BEGIN
ERROR(9)
END;

***********************************************************
( * PROCEDURE EMITCH PUTS THE CHARACTER GIVEN AS A PARAMETER * )
( * INTO THE OBJECT FILE AND UPDATES THE CHECKSUM. CHARACTERS * )
( * 0 THRU 9 HAVE VALUES 0 TO 9, AND A THRU Z HAVE VALUES 10 * )
( * TO 35, ALL OTHER CHARACTERS HAVE VALUE ZERO. * )
*****************************************************************************
PROCEDURE EMITCH ( C : CHAR);
BEGIN
  WRITE(OBJ,C);
  IF C IN ['0'..'9'] THEN CKSUM := CKSUM + ORD(C) - ORD('0');
  IF C IN ['A'..'Z'] THEN CKSUM := CKSUM + ORD(C) + 10 - ORD('A');
  OBJCNT := OBJCNT + 1
END;
*****************************************************************************
( * PROCEDURE EMIT PUTS A BYTE INTO THE OBJECT FILE, UPDATES * )
( * THE CHECKSUM AND INCREMENTS THE PROGRAM COUNTER BY 1. * )
*****************************************************************************
PROCEDURE EMIT(DATA : INTEGER);
VAR 1 : INTEGER;
BEGIN
  DATA := DATA MOD 256;
  IF DATA < 0 THEN DATA := DATA + 256;
  IF PASS2 THEN BEGIN
    IF HEXLOC > HEXMAX-2 THEN BEGIN
      IF LINECNT > LMAX THEN ZPAGE;
      FOR I := 1 TO DWIDTH DO
        BEGIN
          WRITE(LST,OLINE'I1);
          OLINE'I1 := '1 ;
        END;
        WRITE(LST);
        LINECNT := LINECNT + 1;
        HEXLOC := 11;
        END;
        OLINE'HEXLOC1 := HEX(DATA DIV 16);
        OLINE'HEXLOC11 := HEX(DATA);
        HEXLOC := HEXLOC + 2;
        IF OBJCNT > OBJMAX THEN BEGIN
          OBJCNT := 0;
          WRITE(OBJ,'&',HEX(CKSUM DIV 16),HEX(CKSUM));
          CKSUM := 0
          END;
          IF OBJCNT=0 THEN BEGIN
            IF ASWTYPE=ABSOLUTE THEN BEGIN
              WRITE(OBJ,HEX(PC DIV 4096),HEX(PC DIV 256),
              HEX(PC DIV 16),HEX(PC));
              CKSUM := PC + PC DIV 256
            END ELSE BEGIN
              WRITE(OBJ,HEX);
PROCEDURE EMIT(VALUE : INTEGER; ADM : ADRTYPE);
VAR SYM : ARRAY[1..SMAX] OF CHAR;
I : INTEGER;
BEGIN
VALUE := VALUE MOD 65536;
IF VALUE < 0 THEN VALUE := VALUE + 55536;
HEXLOC := HEXLOC+1;
IF ADM IN 'ABSOLUTE, IMMEDIATE, EXTDEF' THEN
BEGIN
EMIT(VALUE);
EMIT(VALUE DIV 256);
END
ELSE BEGIN
IF OBJCNT > OBJMAX-10 THEN BEGIN
OBJCNT := 0;
WRITELN(OBJ, '6', HEX(CKSUM DIV 16), HEX(CKSUM));
END
END
ELSE BEGIN
WRITELN(OBJ, '6', HEX(PC DIV 4096), HEX(PC DIV 256), HEX(PC DIV 16), HEX(PC));
CKSUM := PC + PC DIV 256
END
END
END
END
END
END
END
OBJCNT := OBJCNT+1;
OLINE+HEXLOC := 'R';
HEXLOC := HEXLOC+1;
EMIT(VALUE);
EMIT(VALUE DIV 256);
END
ELSE IF ADR = EXTRF THEN
BEGIN
EMITCH('X');
UNPACK(PEXT,SYM,1):
I := 1;
WHILE (I<=SMAX) AND (SYM"I<>1') DO
BEGIN
EMITCH(SYM"I));
I := I + 1;
END;
EMITCH('X');
IF HEXLOC>HEXMAX-2 THEN
FOR I := 0 TO 3 DO OLINE+HEXLOC+I := 'X';
HEXLOC := HEXLOC + 6;
PC := PC + 2
END
END

******************************************************************
(* THIS ROUTINE CHECKS FOR INDIRECT REFERENCES USING THE IX *)
(* AND IV REGISTERS, AND GENERATES THE PROPER PREFIX IF *)
(* SUCH A REFERENCE IS BEING USED. IT THEN EMITS THE DATA *)
(* BYTE, FOLLOWING IT WITH THE VALUE OF THE DISPLACEMENT *)
(* INDICATED IN THE ASSEMBLED INSTRUCTION. (THIS VALUE IS *)
(* STORED IN GLOBAL VARIABLE 'DBYTE').)
******************************************************************

PROCEDURE EMITIX(data : integer);
BEGIN
IF (IX IN OPSET1) OR (IX IN OPSET2) THEN EMIT(221)
ELSE IF (IY IN OPSET1) OR (IY IN OPSET2) THEN EMIT(253);
EMIT(data);
IF ('IX,IX+OPSET1<>1) OR ('IX,IY+OPSET2<>1) THEN
BEGIN
IF ONLY8080 THEN ERROR(13); 
EMIT(DBYTE)
END
END

******************************************************************
(* THIS ROUTINE MERELY EMITS 'ED' HEXADECIMAL FOLLOWED BY *)
(* THE INPUT DATA VALUE. THIS IS ONLY A CONVENIENCE. *)
******************************************************************

PROCEDURE EMITED (data : integer);
BEGIN
IF ONLY8000 THEN ERROR(13);
EMIT(237);
EMIT(DATA)
END:

**************************************************************************
(* THIS ROUTINE CHECKS FOR USE OF IX AND IY IN A NON-INDIRECT
FASHION. IF THIS IS FOUND, THE PROPER PREFIX IS GENERATED)
(* DBYTE IS NOT EMITTED.
**************************************************************************

PROCEDURE EMITXY0 (DATA : INTEGER);
BEGIN
IF (RX IN OPSET1) OR (RX IN OPSET2) THEN
BEGIN
IF ONLY8000 THEN ERROR(13);
EMIT(221);
END
ELSE IF (RY IN OPSET1) OR (RY IN OPSET2) THEN
BEGIN
IF ONLY8000 THEN ERROR(13);
EMIT(253);
END;
EMIT(DATA)
END:

**************************************************************************
(* PROCEDURE OUTESD RECURSIVELY SEARCHES THE SYMBOL TABLE
FOR ALL EXTERNAL DEFINITIONS AND REFERENCES. IT THEN
WRITES THESE REFERENCES INTO THE OBJECT FILE IN AN 'S'
BLOCK(S).
**************************************************************************

PROCEDURE OUTESD (PTR : STPTR);
PROCEDURE OUTX(N : ALFA; A : ADTYPE; V : INTEGER);

VAR C : CHAR;
NCHAR : ARRAY[1..101] OF CHAR;
I : INTEGER;
BEGIN
IF OBJCNT>OBJMAX-16 THEN
BEGIN
WRITELN(OBJ,'8',HEX(CKSUM DIV 16),HEX(CKSUM));
OBJCNT := 0;
END;
IF OBJCNT=0 THEN
BEGIN
WRITE(OBJ,HASH);
CKSUM := 0;
EMITCH('S');
END;
CASE A OF
CASE A OF
EXTREF : C := 'F';
EXTERNAL C := 'A';
EXTERNAL C := 'R';
END;
EMITCH(C);
UNPACK(N,NCHAR,1);
I := 1;
WHILE (I<SMAX) AND (NCHAR'I!'<>') DO
BEGIN
EMITCH(NCHAR'I!');
I := I + 1
END;
EMITCH('!');
IF (A=EXTERNAL) OR (A=EXTERNAL) THEN EMITW(V,ABSOLUTE);
END;
BEGIN
WITH PTR- DO BEGIN
IF LSON <> NIL THEN OUTESD(LSON);
IF ADMODE IN 'EXTERNAL,EXTEGR,EXTREF' THEN
OUTX(NAME,ADMODE,VALUE);
IF RSON <> NIL THEN OUTESD(RSON)
END;
END.

END

BEGIN
OLINE+HEXLOC! := HEX(PC DIV 4096);
OLINE+HEXLOC+11 := HEX(PC DIV 256);
OLINE+HEXLOC+21 := HEX(PC DIV 16);
OLINE+HEXLOC+31 := HEX(PC);
HEXLOC := HEXLOC + 6
END;

BEGIN
PROCEDURE INLINE READS A LINE FROM THE INPUT FILE AND
PUTS IT IN GLOBAL BUFFER ARRAY INP. IF NO NON-BLANK
CHARACTERS WERE ENCOUNTERED, GLOBAL BOOLEAN NOTBLANK IS
FALSE AT EXIT.

BEGIN
VAR I : INTEGER;
BEGIN
EPPOS := HEXMAX+1;
FOR i := 1 TO IWIDTH DO INP'I! := ' ';  
I := 0;
STOP := EOF(SOURCE);
NOTBLANK := FALSE;
WHILE (I<IWIDTH) AND NOT STOP DO
BEGIN
I := I + 1;
IF EOLN(SOURCE) THEN
BEGIN
READLN(SOURCE);
I := IWIDTH + 1
END
ELSE BEGIN
READ(SOURCE, INP'I');
IF INP'I' <> ' ' THEN NOTBLNK := TRUE;
IF (I = IWIDTH) THEN READLN(SOURCE)
END
END;

{*******************************************************************************}
(* THIS PROCEDURE MOVES THE INPUT POINTER (IPOS) UNTIL A *)
(* NON-BLANK CHARACTER IS ENCOUNTERED, OR THE END IS REACHED *)
{*******************************************************************************}

PROCEDURE SKPBLNK:
BEGIN
WHILE (INP'IPOS = ' ') AND (IPOS < IWIDTH) DO IPOS := IPOS + 1
END;

{*******************************************************************************}
(* PROCEDURE GETOP EVALUATES ALL OPERANDS EXCEPT STRINGS *)
(* ENCLOSED IN PARENTHESES, AN INDIRECT REFERENCE IS *)
(* INDICATED. *)
(* INTERNAL PROCEDURES EVAL1 AND EXPR ARE USED IN EVALUATING*)
(* THE OPERAND.*)
{*******************************************************************************}

PROCEDURE GETOP (VAR OPS : OPSET; VAR VAL : WORD; 
VAR ADM : ADRTYPE);

VAR
INDR : BOOLEAN;

FUNCTION HEXVAL(CH : CHAR) : INTEGER;
BEGIN
IF CH IN '0'..'9' THEN HEXVAL := ORD(CH) - ORD('0')
ELSE IF CH IN 'A'..'F' THEN HEXVAL := ORD(CH) + 10 - ORD('A')
ELSE BEGIN
HEXVAL := 0;
ERROR(3)
END
END;

{*******************************************************************************}
(* EVAL1 EVALUATES ONE FACTOR AT A TIME. IT DOES THE *)
(* SYMBOL TABLE LOOKUP, CHARACTER-TO-INTEGER CONVERSIONS *)
PROCEDURE EVALLVAR VAR VAL : INTEGER; VAR OPS : OPSET;
VAR ADM : ADRTYPE;

VAR
SYM : ARRAY[1..SMAX] OF CHAR;
PSYM : ALFA;
I,J : INTEGER;
UNARYM : BOOLEAN;

PROCEDURE REFLIST;
VAR
NOMATCH : BOOLEAN;
REFL,LASTREF,NEWREF : REFLINK;
BEGIN
WITH STFOUND DO
IF REFS=NIL THEN
BEGIN
NEW(NEWREF):
NEWREF-.ADDR := PC;
NEWREF-.NEXT := NIL;
REFS := NEWREF
END
ELSE BEGIN
REFL := REFS;
NOMATCH := TRUE;
WHILE (REFL<>NIL) AND NOMATCH DO
BEGIN
NOMATCH := REFL-.ADDR<>PC;
LASTREF := REFL;
REFL := REFL-.NEXT
END;
IF NOMATCH THEN
BEGIN
NEW(NEWREF);
NEWREF-.ADDR := PC;
NEWREF-.NEXT := NIL;
LASTREF-.NEXT := NEWREF
END
END
BEGIN
OPS := "!"
ADM := NULL;
PFDFOUND := 0;
UNARYM := FALSE;
IF INP.IPDS = "('" THEN
BEGIN
INDCT := TRUE;
IPDS := IPDS + 1
END
END:

IF INP'IPOS'='-' THEN
BEGIN
UNARYM := TRUE;
IPOS := IP0S + 1
END:
FOR I := 1 TO SMAX DO SYM'I' := '1';
I := 0;
WHILE (I<SMAX) AND (IP0S<1\WIDTH) AND
NOT ((INP'IPOS' IN ' , ', ',', ',' )' \',-', ', ','/', ','@', ', '[]) DO
BEGIN
I := I + 1;
SYM'I' := INP'IP0S';
IP0S := IP0S + 1
END;
IF SYM'I' IN 'A'..'Z' THEN
BEGIN
IF SYM'I'=QUOTE THEN SYM'I' := '1';
PACK(SYM'I',PSYM);
IF S0RCHST(PSYM,SHEAD) THEN
WITH STFOUND DO
IF PDEF THEN BEGIN
IF INDIRECT THEN PFOUND := VALUE+7
ELSE PFOUND := VALUE;
IF (NOT SPECIAL AND P2XREF) THEN REFLIST;
OPS := PDSET\PFOUND;
VAL := PDVAL\PFOUND;
ADM := REGISTER
END
ELSE BEGIN
IF P2XREF THEN REFLIST;
IF ((PINSTR='EQU' ')')OR(PINSTR='DS'))
AND NOT P2 THEN ERROR(10);
PEXT := PSYM;
VAL := VALUE;
OPS := 'CON';
ADM := ADMODE
END
ELSE BEGIN
ERROR(6);
OPS := 'CON';
VAL := 0;
ADM := ABSOLUTE
END
END
ELSE IF SYM'I' IN '0'..'9' THEN
BEGIN
IF SYM'I' = 'H' THEN
BEGIN
VAL := 0;
FOR J := 1 TO I-1 DO VAL := VAL*16 + HEXVAL(SYM'J')
END
ELSE BEGIN
VAL := 0;
FOR J := 1 TO I DO
BEGIN
VAL := VAL +10 + ORD('SYM')-27;
IF NOT (SYM IN '0'..'9') THEN ERROR(3)
END
OPS := 'CDN';
ADM := IMMEDIATE
END ELSE IF SYM = 'QUOTE THEN
BEGIN
IF (1=1) THEN
BEGIN
SYM := INP; IPOS := IPOS + 2
END;
VAL := ASC-ORD(SYM)!!;
OPS := 'CDN';
ADM := IMMEDIATE
END ELSE IF SYM = '$' THEN
BEGIN
VAL := PC;
OPS := '$RPC$';
ADM := ASMTYPE
END ELSE ERROR(4);
IF UNARYM THEN VAL := VAL;
VAL := VAL MOD 65536;
IF VAL<0 THEN VAL := VAL + 65536;
END;

*****************************************************************************
* THIS FUNCTION EVALUATES EXPRESSIONS IN THE OPERAND *
* FIELD. SINCE PARENTHESES INDICATE INVERSION, THEY *
* ARE NOT ALLOWED IN EXPRESSIONS. THE OPERATORS IN *
* DECREASING ORDER OF PRECEDENCE ARE: (ROWS ARE EQUAL) *
* MODULO('X'), DIVISION('Y'), MULTIPLICATION('X') *
* PLUS('X'), MINUS('Y') *
* AND('X'), OR('Y') *
* ADDRESS MODES ARE NOT PERMITTED TO BE MIXED *
* INDISCRIMINATELY. RELATIVE AND EXTREFS MODES CAN ONLY BE *
* OPERATED ON BY PLUS AND MINUS. NO OPERATIONS ARE *
* ALLOWED ON EXTREF'S. *
*****************************************************************************
FUNCTION EXPR (VALO : INTEGER; VAR ADM : ADRTYPE) : INTEGER;

VAR
VSTACK,OSTACK : INTEGER;
VALUES : ARRAY 1..MSTACK1 OF INTEGER;
OPERATORS : ARRAY 1..MSTACK1 OF OPERS;
ADMMODES : ARRAY 1..MSTACK1 OF ADRTYPE;
TVAL : INTEGER;
TOPS : OPSET;
TADM : ADRTYPE;
PROCEDURE PUSHV(VAL : INTEGER; ADM : ADRTYPE);
BEGIN
  VSTACK := VSTACK + 1;
  VALUES^VSTACK1 := VAL;
  ADMODES^VSTACK1 := ADM
END;

FUNCTION POPV (VAR ADM : ADRTYPE) : INTEGER;
BEGIN
  POPV := VALUES^VSTACK1;
  ADM := ADMODES^VSTACK1;
  VSTACK := VSTACK-1
END;

FUNCTION OPCHAR(CH : CHAR) : OPERS;
BEGIN
  CASE CH OF
    '+' : OPCHAR := PLUS;
    '-' : OPCHAR := MINUS;
    '*' : OPCHAR := ATIMES;
    '/' : OPCHAR := ADIV;
    '%' : OPCHAR := AMDIV;
    '&': OPCHAR := AND;
    '|' : OPCHAR := OR;
  END
END;

PROCEDURE PUSHD (VAL : OPERS);
BEGIN
  OSTACK := OSTACK + 1;
  OPERATORS^OSTACK1 := VAL
END;

FUNCTION POPD : OPERS;
BEGIN
  POPD := OPERATORS^OSTACK1;
  OSTACK := OSTACK - 1
END;

FUNCTION TOPD : OPERS;
BEGIN
  TOPD := OPERATORS^OSTACK1
END;

PROCEDURE PERFORM;
VAR V1, V2, V3 : INTEGER;
  A1, A2 : ADRTYPE;
  OP : OPERS;
BEGIN
  V2 := POPV(A2);
  V1 := POPV(A1);
  OP := POPD;
  CASE OP OF
    PLUS: V1 := V1 + V2;
    MINUS: V1 := V1 - V2;
    ATIMES: V1 := V1 * V2;
    ADIV: V1 := V1 div V2;
    AMDIV: V1 := V1 mod V2;
    AND: V1 := V1 and V2;
    OR: V1 := V1 or V2;
    otherwise: V1 := V1
  END;
END;
MINUS: V1 := V1 - V2;
TIMES: V1 := V1 * V2;
DIV: V1 := V1 DIV V2;
MOD: V1 := V1 MOD V2;
AND: BEGIN AND(V1, V2, V3): V1 := V3; END;
OR: BEGIN OR(V1, V2, V3): V1 := V3; END;
END;
IF ('REGISTER, EXREF{A1, A21}+1) OR
( (RELATIVE IN 'A1, A21) AND (OP IN 'ATIMES, 'DIV, 'MOD, 'AND, 'OR))
THEN BEGIN
  ERROR(7);
  A1 := NULL
END
ELSE IF RELATIVE IN 'A1, A21 THEN A1 := RELATIVE
ELSE IF ABSOLUTE IN 'A1, A21 THEN A1 := ABSOLUTE
ELSE IF EXTDEFR IN 'A1, A21 THEN A1 := EXTDEFR
ELSE IF EXTDEFA IN 'A1, A21 THEN A1 := EXTDEFA
ELSE IF IMMEDIATE IN 'A1, A21 THEN A1 := IMMEDIATE
ELSE ERROR(8);
PUSHV(V1, A1)
END;
BEGIN
VSTACK := 0;
OSTACK := 0;
PUSHV(VAL0, ADM);
PUSHQ(NOOP);
WHILE INP•'IPOS IN '•', '•', '•', '/', '•', '&', '•', '•' DO
  BEGIN
    WHILE PRIORITY•OPCHAR(INP•'IPOS) < PRIORITY•TOPPOP DO
      PERFORM;
      OPCHAR(INP•'IPOS));
      IPOS := IPOS + 1;
      EVAL1(TVAL, TOPS, TADM);
      PUSHV(TVAL, TADM);
      END;
    WHILE PRIORITY•TOPPOP > 0 DO PERFORM;
    TVAL := POPV(ADM) MOD 65536;
    IF TVAL < 0 THEN TVAL := TVAL + 65536;
    EXPR := TVAL
  END;
BEGIN
SKPB
INDIRCT := FALSE;
IF INP•'IPOS = 1 THEN IPOS := IPOS + 1;
EVAL1(VAL, OPS, ADM);
IF •IX, IY1•OPS < 1 THEN
  BEGIN
    ADM := RELATIVE;
    DBYTE := EXPR(0, ADM);
    ADM := IMMEDIATE
  END
ELSE VAL := EXPR(VAL, ADM);
IF (CON IN OPS) AND INDRCT THEN OPS := 'ICON1';
IF INDRCT THEN IPOS := IPOS + 1
END;

*******************************************************************************
(  PROCEDURE PASS IS THE HEART OF THE ASSEMBLER. IT CONTAINS)
(  THE BASIC LOOP FOR SCANNING THE INPUT AND CO-ORDINATING *)
(  THE OTHER PROCEDURES. THE FIRST OPERATION IS TO GET A LINE
OF TEXT. IF BOOLEAN VARIABLE INMACRO IS TRUE, THE LINE IS)
(  GENERATED BY PROCEDURE MACLINE (IMPLEMENTED ONLY AS AN ERROR
CALL IN V1.0). THE BEGINNING-OF-LINE SYMBOL (IF ANY) IS *)
(  COLLECTED INTO VARIABLE PSYM, AND THE INSTRUCTION *)
(  MNEMONIC IS PUT INTO VARIABLE PINSTR. IF THE INSTRUCTION*)
(  IS FOUND IN THE OPCODE TABLE, PROCESSING BEGINS. IF NOT,*)
(  THE SYMBOL IS PASSED TO PROCEDURE GETMAC, WHICH WILL LOOK*)
(  THE SYMBOL UP IN A MACRO TABLE. (GETMAC IS ONLY AN ERROR *)
(  CALL IN V1.0) FIELD NOPNOS OF THE FOUND OPCODE TABLE *
(  ELEMENT TELLS HOW MANY OPERANDS TO EVALUATE BEFORE STARTING
(  THE INSTRUCTION-PROCESSING CASE STATEMENT. FIELD SWTCH *)
(  IS THE VALUE USED TO JUMP INTO THE CASE STATEMENT. *)
(  THE CASE STATEMENT ITSELF IS MERELY A COLLECTION OF TESTS*)
(  TO SEE IF THE OPERANDS FALL INTO CERTAIN CLASSES FOR THE *)
(  PARTICULAR INSTRUCTION. CODE SEQUENCES ARE THEN GENERATED
(  WHEN AN ACCEPTABLE COMBINATION OF OPERANDS IS FOUND. *)
(  THE LOOP IS ENDED WITH THE PRINT STATEMENT WHICH IS
(  EXECUTED ONLY IF IT IS PASS2 AND THE NOT-SUPPRESS FLAG IS*
(  TRUE. THUS SPEAKE ZARATHUSTRA *)
*******************************************************************************

PROCEDURE PASS (PARAM: INTEGER):
VAR
I,J: INTEGER; '*
ADM1,ADM2: ADTYPE;
FUNCTION STRING: INTEGER;
VAR I : INTEGER;
EXIT : BOOLEAN;
BEGIN
I := 0;
IPOS := IPOS + 1;
EXIT := FALSE;
REPEAT
   IF IPOS > IWIDTH THEN
      BEGIN
         EXIT := TRUE;
         ERROR(5);      
      END
   ELSE IF INP>IPOS <> SQQUOTE THEN
      BEGIN
         I := I + 1;
         STR'I := INP'IPOS;
         IPOS := IPOS + 1
END
ELSE IF (IPOS<WIDTH) AND (INP'IPOS+11=SQUOTE) THEN
BEGIN
I := I + 1;
STR'I1 := SQUOTE;
IPOS := IPOS + 2
END
ELSE EXIT := TRUE
UNTIL EXIT;
IPOS := IPOS + 1;
STRING := 1
END;

PROCEDURE IMCDEC(DEFLT2 : BYTE):
BEGIN
IF RB IN OPSET1 THEN EMIT(DEFLT + VAL1*8)
ELSE IF 'RBD,RH,RSP1+OPSET1<>1 THEN EMIT(DEFLT2 + VAL1*16)
ELSE IF 'IX,IY1+OPSET1<>1 THEN EMITXY(DEFLT+48)
ELSE IF 'RX,RY1+OPSET1<>1 THEN EMITXY0(DEFLT2 + 32)
ELSE ERROR(4)
END;
BEGIN
PASS1 := PARAM = 1;
PASS2 := PARAM = 2;
P2XREF := PASS2 AND NOT NOXREF;
LINECNT := 100;
INMACRO := FALSE;
STOP := FALSE;
NOTEND := TRUE;
ASMTYPE := ABSOLUTE;
RELSAVE := 0;
NOXREF := TRUE;
LIST := TRUE;
SYMTAB := TRUE;
PC := 0;
ERRCNT := 0;
PGCNT := 1;
RESET(SOURCE);
REPEAT
IF INMACRO THEN MACLINE
ELSE INLINE;
DBYTE := 0;
NOSUPRES := TRUE;
FOR I := 1 TO OFFSET DO OLINE'I1 := ' ';
OLINE'OFSYTE := COLON;
FOR I := 1 TO WIDTH DO
OLINE'I1+OFFSET1 := INP'I1;
FOR I := 1 TO SMAX DO SYMB'I1 := ' ';
IPOS := 1;
IF ((INP'I1 <> ' ') AND (INP'I1 <> ' ')) AND NOTBLANK THEN
BEGIN
(* START LINE PROCESSING *)
WHILE (IPOS<SMAX+1) AND (INP'IPOS <> ' ') DO
BEGIN
(* COLLECT SYMBOL *)
SYMBOL*IPOSI := INP*IPOSI;  (* AT START OF *)
IPOS := IPOS + 1  (* LINE, IF ANY *)
END;
Packing SYMBOL,PSYM);
(* NOW SKIP REST OF LONG SYMBOL *)
WHILE INP*IPOSI <> ' ' DO IPOS := IPOS + 1;
SKIPBLNK;
FOR I := 1 TO SMAX DO INSTR*II := ' ';
I := IPOS;  
WHILE INP*IPOSI <> ' ' DO
BEGIN
INSTR*IPOSI := INP*IPOSI;
IPOS := IPOS + 1
END;
Packing(INSTR,1,PINSTI);
IF NOT SRCQOT(PINSTR,OTHERHEAD) THEN GETMAC
ELSE BEGIN
DEFRT := OFFOUND-.DFLT;
IF OFFOUND-,NOPDOS > 0 THEN GETOP(OPSET1,VAL1,ADM1);
IF OFFOUND-,NOPDOS = 2 THEN GETOP(OPSET2,VAL2,ADM2);
HEXLOC := 5;
IF OFFOUND-,SWITCH IN '1..23,28..30' THEN
BEGIN
ADDRESS:
IF PSYM<>' ' THEN INSSST(PSYM,PC,FALSE,ASMTYPE,FALSE)
END;
CASE OFFOUND-,SWITCH OF
(* 1-BYTE, NO-OPERAND INSTRUCTIONS *)
1: EMT(DEFRT);
(* 2-BYTE, NO-OPERAND INSTRUCTIONS *)
2: EMTED(DEFRT);
(* ADC INSTRUCTION *)
3: IF RA IN OPSET1 THEN
   IF RB IN OPSET2 THEN
      EMT(136+VAL2)
   ELSE IF ((IX,YI)*OPSET2<>'1') THEN
      EMTXY(142)
   ELSE IF C0N IN OPSET2 THEN
      BEGIN
         EMT(206);
         EMT(VAL2)
      END
   ELSE ERROR(4)
   ELSE IF (RH IN OPSET1) AND ((RBD,RSP)+OPSET2<>'1') THEN
      EMITED(74+VAL2+16)
   ELSE ERROR(4);
(* ADD INSTRUCTION *)
4: IF RA IN OPSET1 THEN
   IF RB IN OPSET2 THEN
      EMIT(128+VAL2)
      ELSE IF 'IX, IY1+OPSET2<>1 THEN
         EMITY(134)
      ELSE IF CON IN OPSET2 THEN
         BEGIN
         EMIT(198);
         EMIT(VAL2)
         END
      ELSE ERROR(4)
   ELSE IF (RH IN OPSET1) AND ('RBD, RH, RSP1*OPSET2<>1) THEN
      EMIT(9 + VAL2*16)
   ELSE IF('RX, RY1+OPSET1<>1) AND ('RBD, RX, RY, RSP1*OPSET2<>1)
      THEN EMITTO(9 + VAL2*16)
      ELSE ERROR(4);
      (* AND, OR, XOR, CP, & SUB INSTRUCTIONS *)

5: IF RB IN OPSET1 THEN
   EMIT(DEFLT + VAL1)
   ELSE IF 'IX, IY1+OPSET1<>1 THEN
      EMITY(DEFLT + 6)
   ELSE IF CON IN OPSET1 THEN
      BEGIN
      EMIT(DEFLT + 70);
      EMIT(VAL1)
      END
   ELSE ERROR(4);
      (* BIT, SET, & RES INSTRUCTIONS *)

6: BEGIN
   IF ONLY8080 THEN ERROR(13);
   IF CON IN OPSET1 THEN
   IF RB IN OPSET2 THEN
      BEGIN
      EMIT(203);
      EMIT(DEFLT + VAL2 + VAL1*8)
      END
   ELSE IF 'IX, IY1+OPSET2<>1 THEN
      BEGIN
      EMITY(203);
      EMIT(DEFLT + 6 + VAL1*8)
      END
   ELSE ERROR(4)
   ELSE ERROR(4);
   END;
      (* CALL INSTRUCTION *)

7: IF CON IN OPSET1 THEN
   BEGIN
   EMIT(205);
   EMITW(VAL1, ADH1)
END
ELSE IF CF IN OPSET1 THEN
BEGIN
IF RB IN OPSET1 THEN VAL1 := 3;
GETOP(OPSET2,VAL2,ADM2);
IF CON IN OPSET2 THEN
BEGIN
EMIT(196 + VAL1*8);
EMITW(VAL2,ADM2)
END
ELSE ERROR(4)
END
ELSE ERROR(4);

(* RET INSTRUCTION *)

8: BEGIN
SKPBLNK;
OPSET1 := '1';
IF IP<WIDTH THEN
BEGIN
FOR I := 1 TO SMAX DO SYMBOL[I] := '1';
I := 1;
WHILE ((I<4)AND(IP<WIDTH))AND(IP<IP+1) DO
BEGIN
SYMBOL[I] := IP<IP+1;
I := I+1;
IP := IP+1;
END;
PACK(SYMBOL,PSYM);
IF SRCST(PSYM,STHEAD) THEN
WITH STFOUND- DO
IF PDEF THEN
BEGIN
OPSET1 := PDEF<VAL1;
VAL1 := PDVAL<VAL1;
END
END:
IF OPSET1='1' THEN
EMIT(201)
ELSE IF CF IN OPSET1 THEN
BEGIN
IF RB IN OPSET1 THEN VAL1 := 3;
EMIT(192 + VAL1*8)
END
ELSE ERROR(4)
END:

(* JP INSTRUCTION *)

9: IF CON IN OPSET1 THEN
BEGIN
EMIT(195);
EMITW(VAL1,ADM1)
END
ELSE IF CF IN OPSET1 THEN
  BEGIN
  IF RB IN OPSET1 THEN VAL1 := 3;
  GETOP(OPSET2, VAL2, ADM2);
  EMIT(194 + VAL1+8);
  EMITw(VAL2, ADM2)
  END
ELSE IF 'IH, IX, IY' OPSET1<>'I THEN
  BEGIN
  IF IV IN OPSET1 THEN
  BEGIN
  IF ONLY8080 THEN ERROR(13);
  EMIT(253)
  END
ELSE IF IX IN OPSET1 THEN
BEGIN
  IF ONLY8080 THEN ERROR(13);
  EMIT(221)
END;
EMIT(233)
END
ELSE ERROR(4):
(* JR INSTRUCTION *)
10: BEGIN
  IF ONLY8080 THEN ERROR(13);
  IF CON IN OPSET1 THEN
BEGIN
  EMIT(24):
  EMIT(v+1-2):
  IF (VAL1<2) OR ((VAL1 >129) AND (VAL1<65410)) THEN ERROR(11)
  END
ELSE IF CF1 IN OPSET1 THEN
BEGIN
  IF RB IN OPSET1 THEN VAL1 := 3;
  GETOP(OPSET2, VAL2, ADM2);
  EMIT(32 + VAL1+8);
  EMIT(VAL2-2):
  IF (VAL2<2) OR ((VAL2 >129) AND (VAL2<65410)) THEN ERROR(11)
  END
ELSE ERROR(4):
END;
(* DEC INSTRUCTION *)
11: INCDEC(11):
(* INC INSTRUCTION *)
12: INCDEC(3):
(* EX INSTRUCTION *)
13: IF (ISP IN OPSET1) AND (\'RH,RX,RY\'\*OPSET2<>\'I\') THEN
   EMITXYO(227)
ELSE IF (RAF IN OPSET1) AND (RAF IN OPSET2) THEN
   BEGIN
      IF ONLY8080 THEN ERROR(13);
      EMIT(8);
   END
ELSE IF (R80 IN OPSET1) AND (RH IN OPSET2) THEN EMIT(235)
ELSE ERROR(4):
(*) INSTRUCTION *)
14: IF (VAL1<>0) OR (VAL1<>2) THEN ERROR(4)
   ELSE CASE VAL1 OF
      0: EMITED(70);
      1: EMITED(86);
      2: EMITED(94)
   END;
(*) OUT INSTRUCTION *)
15: IF (IC IN OPSET1) AND (R8 IN OPSET2) THEN
    EMITED(65 + VAL2\*8)
ELSE IF (ICON IN OPSET1) AND (RA IN OPSET2) THEN
    BEGIN
      EMIT(211);
      EMITVAL1;
    END
ELSE ERROR(4):
(*) PUSH & POP INSTRUCTIONS *)
16: IF \'RAF,RBD,RH,RX,RY\'\*OPSET1<>\'I\' THEN
    EMITXYO(DEFLT + VAL1\*16)
ELSE ERROR(4):
(*) RL,RLC,RR,RRC,SLA,SRA, & SRL INSTRUCTIONS *)
17: BEGIN
    IF ONLY8080 THEN ERROR(13);
    IF \'R8,RA,HI\'\*OPSET1<>\'I\' THEN EMIT(203)
    ELSE IF \'IX,IY\'\*OPSET2<>\'I\' THEN EMITXY(203)
    ELSE ERROR(4);
    EMIT(DEFLT + VAL1)
   END:
(*) SBC INSTRUCTION *)
18: IF RA IN OPSET1 THEN
   IF R8 IN OPSET2 THEN EMIT(152 + VAL2)
   ELSE IF \'IX,IY\'\*OPSET2<>\'I\' THEN EMITXY(152)
   ELSE IF CON IN OPSET2 THEN
      BEGIN
         EMIT(222);
EMIT(VAL2)
END
ELSE ERROR(4)
ELSE IF (RH IN OPSET1) AND
(RBD,RH,RSPI+OPSET2<>1) THEN
EMITED(66+VAL2*16)
ELSE ERROR(4);
(* NOSYM PSEUDO-OP *)
19:SYMTHB := FALSE;
(* DJNZ INSTRUCTION *)
20:BEGIN
EMIT(16);
EMIT(VAL1-2)
END;
(* IN INSTRUCTION *)
21:IF (R8 IN OPSET1) AND (IC IN OPSET2) THEN
EMITED(64+VAL1*8)
ELSE IF (RA IN OPSET1) AND (ICON IN OPSET2) THEN
BEGIN
EMIT(219);
EMIT(VAL2)
END
ELSE ERROR(4);
(* RST INSTRUCTION *)
22:BEGIN
IF (VAL1 MOD 8 <> 0) OR (VAL1 > 56) THEN ERROR(4):
EMIT(199+VAL1)
END;
(* LD INSTRUCTION *)
23:IF R8 IN OPSET1 THEN
IF R8 IN OPSET2 THEN EMIT(64+VAL2+VAL1*8)
ELSE IF CON IN OPSET2 THEN
BEGIN
EMIT(6+VAL1*8):
EMIT(VAL2)
END
ELSE IF (RA IN OPSET1) AND (ICON IN OPSET2) THEN
BEGIN
EMIT(58):
EMIT(VAL2,ADN2)
END
ELSE IF 'IX,111+OPSET2<11 THEN EMITXX(70+VAL1*8)
ELSE IF (RA IN OPSET1) AND (IBD IN OPSET2) THEN
EMIT(10+VAL2*16)
ELSE IF RIR IN OPSET2 THEN EMITED(87+VAL2)
ELSE ERROR(4)
ELSE IF ('RBD, RX, RY, RH, RSP1*OPSET1<>1) AND (RA IN OPSET2) THEN BEGIN
EMITXYO(1 + VAL1*16);
EMITW(VAL2, ADM2)
END
ELSE IF ICON IN OPSET2 THEN IF RA IN OPSET1 THEN
BEGIN
EMIT(58);
EMITW(VAL2, ADM2)
END
ELSE IF 'RH, RX, RY1*OPSET1<>1 THEN BEGIN
EMITXYO(42);
EMITW(VAL2, ADM2)
END
ELSE IF 'RBD, RSP1*OPSET1<>1 THEN BEGIN
EMITD(75 + VAL1*16);
EMITW(VAL2, ADM2)
END
ELSE ERROR(4)
ELSE IF ICON IN OPSET1 THEN IF 'RH, RX, RY1*OPSET2<>1 THEN BEGIN
EMITXY(34);
EMITW(VAL1, ADM1)
END
ELSE IF 'RBD, RSP1*OPSET2<>1 THEN BEGIN
EMITD(67 + VAL2*16);
EMITW(VAL1, ADM1)
END
ELSE IF RA IN OPSET2 THEN BEGIN
EMIT(50);
EMITW(VAL1, ADM1)
END
ELSE ERROR(4)
ELSE IF 'IX, IY1*OPSET1<>1 THEN IF RB IN OPSET2 THEN EMITXY(112 + VAL2)
ELSE IF ICON IN OPSET2 THEN BEGIN
EMITXY(54);
EMITW(VAL2)
END
ELSE ERROR(4)
ELSE IF (1BD IN OPSET1) AND (RA IN OPSET2) THEN
EMIT(2 + VAL1*16)
ELSE IF (RSP IN OPSET1) AND (RH, RX, RY1*OPSET2<>1) THEN
EMITXYO(249)
ELSE IF (RIR IN OPSET1) AND (RA IN OPSET2) THEN
EMITD(71 + VAL1)
ELSE ERROR(4);
(* EQU PSEUDO-OP *)

24:BEGIN
  IF PDFFOUND<>0 THEN
    IF (PDFFOUND IN '1'..'14') OR (PDFFOUND IN '30'..'31') THEN
      INST(PSYM, PDFFOUND, TRUE, ADM1, FALSE)
    ELSE ERROR(12)
    ELSE INST(PSYM, VAL1, FALSE, ADM1, FALSE);
    I := PC;
    PC := VAL1;
    ADDRESS;
    PC := I
  END;

(* ORG PSEUDO-OP *)

25:BEGIN
  IF PSYM<>'' THEN
    INST(PSYM, VAL1, FALSE, ABSOLUTE, FALSE);
  IF ASMTYPE = RELATIVE THEN RELSAVE := PC;
  IF OBJCNT<>0 THEN
    BEGIN
      WRITELN(OBJ, '&', HEX(CKSUM DIV 16), HEX(CKSUM));
      OBJCNT := 0
    END;
    PC := VAL1;
    ASMTYPE := ABSOLUTE
  END;

(* REL PSEUDO-OP *)

26:BEGIN
  IF ASMTYPE = ABSOLUTE THEN PC := RELSAVE;
  ASMTYPE := RELATIVE;
  IF PSYM<>'' THEN INST(PSYM, PC, FALSE, EXTDEF, FALSE);
  IF OBJCNT<>0 THEN
    BEGIN
      WRITELN(OBJ, '&', HEX(CKSUM DIV 16), HEX(CKSUM));
      OBJCNT := 0
    END;
  END;

(* END PSEUDO-OP *)

27:BEGIN
  STOP := TRUE;
  NOTEND := FALSE;
  FOR I:=1 TO 6 DO HEADER'I' := ''
  END;

(* DB PSEUDO-OP *)

28:REPEAT
  SKPBLNK;
IF INP'IPOS'<', THEN IPOS := IPOS + 1;
IF INP'IPOS'='QUOTE THEN
BEGIN
J := string;
FOR I := 1 TO IP0S DO EMIT(ASC'ORD(STR'J'));
END
ELSE BEGIN
GETOP(OPSET1,VAL1,ADM1);
IF ADM1 IN 'ABSOLUTE,IMMEDIATE' THEN EMIT(VAL1)
ELSE ERROR(4)
END
UNTIL (IPOS=IWIDTH) OR (INP'IPOS'<',');

(* DW PSEUDO-OP *)

29:REPEAT
GETOP(OPSET1,VAL1,ADM1);
EMIT(VAL1,ADM1)
UNTIL (IPOS=IWIDTH) OR (INP'IPOS'<',');

(* DS PSEUDO-OP *)

30:IF CON IN OPSET1 THEN
BEGIN
IF OBJCNT<0 THEN
WRITELN(OBJ,'&',HEX(CKSUM DIV 16),HEX(CKSUM));
OBJCNT := 0;
PC := PC + VAL1
END
ELSE ERROR(4);

(* EXT PSEUDO-OP *)

31:IF SRCHST(PSYM,STHEAD) THEN
BEGIN
IF PASS1 THEN WITH STFOUND= DO
IF ADMODE=RELATIVE THEN ADMODE := EXTDEFR
ELSE ADMODE := EXTDEFA
END
ELSE INST(PSYM,0,FALSE,EXTREF,FALSE);

(* TITLE PSEUDO-OP *)

32:BEGIN
WHILE(IPOS<IWIDTH) AND (INP'IPOS'='QUOTE) DO
BEGIN
IP0S := IP0S + 1;
J := string;
IF J>0 THEN
BEGIN
FOR I := J+1 TO IWIDTH DO STR'J' := '1'
FOR I := 1 TO 6 DO
PACK(STR,(I-1)*10+1,TITLE');
END
END
(* MACRO PSEUDO-OP *)
33:;
(* MEND PSEUDO-OP *)
34:;
(* GEN PSEUDO-OP *)
35:PRGEN := TRUE;
(* NOGEN PSEUDO-OP *)
36:PRGEN := FALSE;
(* LIST PSEUDO-OP *)
37:LIST := TRUE;
(* NOLIST PSEUDO-OP *)
38:LIST := FALSE;
(* XREF PSEUDO-OP *)
39:XXREF := FALSE;
(* PAGE PSEUDO-OP *)
40:BEGIN
   NOSUPRES := FALSE;
   LINCNT := LMAX+10
END;
(* ONLY6080 PSEUDO-OP *)
41:ONLY6080 := TRUE;
(* HEADER PSEUDO-OP *)
42:IF PASS2 THEN BEGIN
   NOSUPRES := FALSE;
   WHILE (IPOS<1WIDTH) AND (INP'IPOS1<>SQUOTE) DO
      IPOS := IPOS + 1;
      J := STRING;
      IF J>0 THEN BEGIN
         FOR I := J+1 TO 1WIDTH DO STR'I1 := '1'
      END;
      FOR I := 1 TO 8 DO PACK(STR,(I-1)+10+1,HEADER'I1)
END;
(* MESSAGE PSEUDO-OP *)
43:IF PASS2 THEN BEGIN
   WHILE (IPOS<1WIDTH) AND (INP'IPOS1<>SQUOTE) DO
      IPOS := IPOS + 1;
      J := STRING;
      WRITE('1');
      IF J>0 THEN BEGIN
         FOR I:=1 TO J DO WRITE(STR'I1);
         WRITE;
      END;
END;
(*) DISPLAY PSEUDO-OP *)
44: IF PASS2 THEN BEGIN
   I := PC; (* SAVE PC *)
   PC := VAL1; (* VALUE TO BE DISPLAYED *)
   ADDRESS; (* DISPLAY VALUE *)
   PC := I; (* RESTORE PC *)
   WRITE(' ');
   WRITE(HEX(VAL1 DIV 4096),HEX(VAL1 DIV 256),HEX(VAL1 DIV 16),HEX(VAL1));
   WRITELN(' HEXADECIMAL');
END;
(*) DUMMY1 *)
45:
(*) DUMMY2 *)
46:
(*) DUMMY3 *)
47:
(*) DUMMY4 *)
48:
END
END: (* END LINE PROCESSING *)
IF (PASS2 AND LIST) AND NOSUPRES THEN
BEGIN
   IF LINECNT > LMAX THEN 2PAGE;
   LINECNT := LINECNT + 1;
   J := UI6WIDTH;
   WHILE 'WJ>1' AND (OLINE[WJ]=' ') DO J := J-1;
   FOR I := 1 TO J DO WRITE(LST,OLINE[I]);
   WRITELN(LST)
   UNTIL STOP;
IF PASS2 AND NOTEND THEN ERROR(2);
IF PASS2 AND (ERRCNT<>0) THEN
BEGIN
   WRITELN(LST,ERRCNT:5,' PROGRAM ERRORS');
   WRITELN(ERRCNT:5,' PROGRAM ERRORS')
END
ELSE IF PASS2 AND (ERRCNT=0) THEN
BEGIN
   WRITELN(LST,' NO PROGRAM ERRORS');
   WRITELN(' NO PROGRAM ERRORS')
END
END:

##########################################################################
BEGIN
STARTIME := CLOCK;
INIT:
PASS(1):
REWITE(OBJ)
OBJCNT := 0;
PAS S2 := TRUE;
OUTESD(STHEAD)
IF OBJCNT<>0 THEN
WITELN(OBJ,'&',HEX(CKSUM DIV 16),HEX(CKSUM));
OBJCNT := 0;
CKSUM := 0;
PAS S(2)
IF OBJCNT<>0 THEN WITELN(OBJ,'&',HEX(CKSUM DIV 16),
HEX(CKSUM));
WITELN(OBJ,'$');
LINECNT := 100;
SPGS := 0;
IF SYMTAG THEN DUMPST(STHEAD)
WITELN(LST)
WITELN(LST,CLOCK-STRTIME:7,' MILLISECONDS USED IN THIS ASSEMBLY');
WITELN(LST,' THIS ASSEMBLY MADE ON ',VDATE,' AT ',VTIME);