AD-A063 004 QUESTRON CORP SAN DIEGO CALIF F/G 9/2
RADIATION HARDENED MICROPROCESSOR. VOLUME II.(U)
MAY 78 V V NICKEL, P A ROSENBERG F33615-77-C-1001
UNCLASSIFIED AFAL-TR-78-55-VOL-2 NL

1 OF 4
AD A063004

(18) (19) AFAL-TR-78-55 – VOL-2
Volume II

LEV

A0630

(6) RADIATION HARDENED MICROPR
Volume II.

(10) V. V. /Nicka
P. A. /Rosen

QUESTRON CORPORATION
SAN DIEGO, CALIFORNIA 92108

(11) MAY 1978

(12) 206 p.

TECHNICAL REPORT AFAL-TR-78-55, Volume II
Final Report for Period Octo[redacted]976 – Dece[redacted]977

(9)

(15) F33615-77-C-

Approved for public release; distribution

AIR FORCE AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORI
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

409 170

79

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Gary D. Gaugler
Project Officer

Stanley E. Wagner, Chief
Microelectronics Branch
Electronic Technology Division

FOR THE COMMANDER

William J. Edwards, Director
Electronic Technology Division

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFAL/DHE ,*W-PAFB, OH 45433 to help us maintain a current mailing list".*

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

AIR FORCE/56780/13 December 1978 — 75

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFAL-TR-78-55 Volume II	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) RADIATION HARDENED MICROPROCESSOR		5. TYPE OF REPORT & PERIOD COVERED Final Oct 76-Dec 77 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) V. V. Nickel, P. A. Rosenberg		8. CONTRACT OR GRANT NUMBER(s) F33615-77-C-1001
9. PERFORMING ORGANIZATION NAME AND ADDRESS Questron Corporation 2727 Camino Del Rio S., Suite 125 San Diego, California 92108		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS CDRL 003 6096-40-04
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Avionics Laboratory Air Force Systems Command, U.S. Air Force Wright-Patterson AFB, Ohio 45433		12. REPORT DATE May, 1978 13. NUMBER OF PAGES 296
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

8080 Microprocessor
Automatic Test Generation
CMOS/SOS
Emulation
General Processor Unit (GPU)
Low Cost, Functional Testing
LSI
LSI Testing
Microprogramming
Radiation Hardened Microprocessor
Universal Array (UA)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Functional capabilities of a CMOS/SOS LSI device family for emulation usage were assessed. The device family consists of an 8-bit processor bit slice called the general processor unit (GPU), a 1024-bit read only memory (ROM) and a Universal Array (UA) device with approximately 300 gates. The devcies were analyzed in detail and deficiencies in their designs are itemized. Recommendations for changes are made to improve the emulation capabilities of the device family. Radiation hardening characteristics of the GPU,

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ROM and GUA are assessed in relation to their usage in emulating existing microcomputers and mini-computers. A complete functional specification is provided for the GPU. During this effort, the GPU design was verified utilizing low cost, functional testing techniques developed specially for LSI design validation. The key to low cost, functional testing is the automatic generation of test vectors. Several faults were detected in the GPU and are itemized in the report. A top level throughput characterization was also performed on the GPU. To the emulation capabilities of the devices, a design of an 8080 microprocessor emulator, utilizing the CMOS/SOS device family, was completed during this effort. Hardware and firmware requirements for the emulator are determined to measure emulation efficiency and a throughput comparison is made between the emulator and the actual 8080.

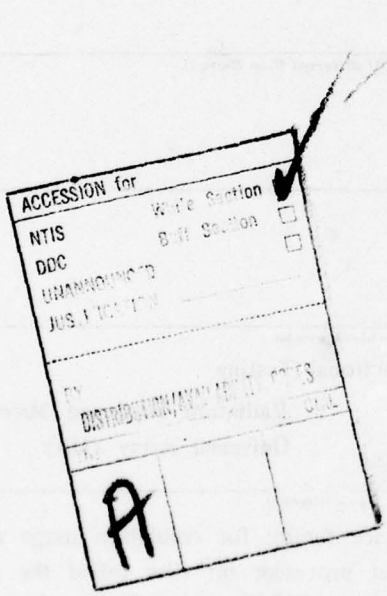


Table of Contents Volume II

5. Support Software Design Description 1
5.1. Overview 1
5.1.1 Q-80 Microprogram Assembler 2
5.1.1.1 User's Reference 2
5.1.1.2 Q-80 Microprogram Assembler Computer Program Documentation 25
5.1.1.2.1 The Main Program MICROX 29
5.1.1.2.1.1 Declaration of external Global Procedures 30
5.1.1.2.1.2 Declaration of Global Variables 30
5.1.1.2.1.3 Declaration of Internal Global Procedures 30
5.1.1.2.1.4 Declaration of External Procedures PASS 1 and PASS 2 30
5.1.1.2.1.5 Prologue Activities Block 31
5.1.1.2.2 Subroutine PASS1 33
5.1.1.2.2.1 Declarations of External Global Procedures 33
5.1.1.2.2.2 Declaration of External Variables and Constants 34
5.1.1.2.2.3 Declaration of Global Procedures 34
5.1.1.2.2.4 Subroutine PASS1 34
5.1.1.2.2.4.1 Declaration of Variables Local to PASS1 35
5.1.1.2.2.4.2 Declaration of Subroutines Local to PASS1 35
5.1.1.2.2.4.2.1 Subroutine BUFFADV 35
5.1.1.2.2.4.2.2 Subroutine NEWWORD 36
5.1.1.2.2.4.2.2.1 Subroutine CODEADV 36
5.1.1.2.2.4.2.2.2 The Body of Subroutine NEWWORD 36
5.1.1.2.2.4.2.3 Subroutine EXTRACTLABEL 38
5.1.1.2.2.4.3 Initializations 41
5.1.1.2.2.4.4 Code to Build Symbol Table 41
5.1.1.2.2.4.4.1 Build Symbol Table 41
5.1.1.2.2.4.4.2 Abort if any Errors Detected 42
5.1.1.2.2.4.4.3 Check Correctness of the Symbol Table 42
5.1.1.2.2.4.4.4 Abort if any Errors Found 43
5.1.1.2.2.4.4.5 Compress Symbol Table 43
5.1.1.2.2.4.4.6 Assignment of Addresses 43
5.1.1.2.2.4.4.7 Allocate Space and Write Address on Disk 43
5.1.1.2.2.4.4.8 Symbol Table Listing 44
5.1.1.2.2.4.4.9 Reopen Source and Binary Files 45
5.1.1.2.3 Subroutine PASS2 45
5.1.1.2.3.1 Declarations of External Global Procedures 45
5.1.1.2.3.2 Declaration of External Variables and Constants 45

5.1.1.2.3.3 Declaration of Global Procedures 46
5.1.1.2.3.4 Subroutine PASS2 46
5.1.1.2.3.4.1 Declaration of Variables Local to PASS2 48
5.1.1.2.3.4.2 Procedures Internal to PASS2 49
5.1.1.2.3.4.2.1 Subroutine BUFFADV 49
5.1.1.2.3.4.2.2 Subroutine NEWWORD 50
5.1.1.2.3.4.2.2.1 Subroutines Internal to Subroutine NEWWORD 50
5.1.1.2.3.4.2.2.1.1 Subroutine LINEOUT 50
5.1.1.2.3.4.2.2.1.2 Subroutine CODEADV 51
5.1.1.2.3.4.2.2.2 The Body of Subroutine NEWWORD 51
5.1.1.2.3.4.2.3 Subroutine NEWPHRASE 53
5.1.1.2.3.4.2.4 Subroutine IDENTIFY 55
5.1.1.2.3.4.2.5 ASCII2BIN Function 55
5.1.1.2.3.4.2.6 Subroutine PRINTROUTINE 56
5.1.1.2.3.4.3 Initializations at Entry into PASS2 56
5.1.1.2.3.4.4 The Microcode Translation Loop 56
5.1.1.2.3.4.4.1 Call to Subroutine NEWWORD 58
5.1.1.2.3.4.4.2 Phrase Translation Loop 58
5.1.1.2.3.4.4.2.0 The Case TYPE=0: Translation of GOTO-phrases 58
5.1.1.2.3.4.4.3.1 The Case TYPE=1: Translation of Register Assignment Phrases . . . 61
5.1.1.2.3.4.4.3.2 The Case TYPE=2: Translation of P1B Phrases 63
5.1.1.2.3.4.4.3.3 The Case TYPE=3: Translation of P2B-Phrases 65
5.1.1.2.3.4.4.3.4 The Case TYPE=4: Translation of DO-Phrases 66
5.1.1.2.3.4.4.3.5 The Case TYPE=5: Translation of ALC-Phrases 67
5.1.1.2.3.4.4.3.6 The Case TYPE=6: Translation of SHIFTER-Phrases 72
5.1.1.2.3.4.4.3.7 The Case TYPE=7: Translation of MXL0OUT-Phrases 73
5.1.1.2.3.4.4.3.8 The Case TYPE=8: Translation of MXL1OUT-Phrases 73
5.1.1.2.3.4.4.3.9 The Case TYPE=9: Translation of MXH0OUT-Phrases 74
5.1.1.2.3.4.4.3.10 The Case TYPE=10: Translation of MXH1OUT-Phrases 74
5.1.1.2.3.4.4.3.11 The Case TYPE=11: Translation of CO-Phrases 75
5.1.1.2.3.4.4.3.12 The Case TYPE=12: Translation of MFLAGS-Phrases 75
5.1.1.2.3.4.4.3.13 The Case TYPE=13: Translation of COUT-Phrases 76
5.1.1.2.4.3.3.4.14 The Case TYPE=14: Translation of the NOLOAD Phrase 76
5.1.1.2.3.4.4.3.15 The Case TYPE=15: Translation of I/O-Phrases 77
5.1.1.2.3.4.4.3.16 The Case TYPE=16: Translation of GUA Phrases 77
77

5.1.1.2.3.4.4.3.17 The Case TYPE=17: Translation of E-Phrases 79
5.1.1.2.3.4.4.3.18 The Case TYPE=18: Errors 79
5.1.1.2.3.4.4.3.19 The Case TYPE=19: End of Microword 79
5.1.1.2.3.4.4.3.20 The Case TYPE=20: Other Phrases 79
5.1.1.2.3.4.4.4 Delayed Resolution of Control Bits 80
5.1.1.2.3.4.4.4.1 Delayed Resolution of S_1 and of the A-Bits 80
5.1.1.2.3.4.4.4.2 Delayed Resolution of M-Bits 81
5.1.1.2.3.4.4.5 Printing and Output of the Translated Microword 83
5.1.2 Q-80 Simulator 174
5.1.2.1 User's Reference 174
5.1.2.2 Q-80 Simulator Computer Program Documentation 189
5.1.2.2.1 Declaration of External Global Procedures 194
5.1.2.2.2 Declaration of Global Constants and Variables 195
5.1.2.2.3 Declaration of Internal Global Procedures 195
5.1.2.2.4 The "Prologue Activities" Block 196
5.1.2.2.5 The "Read Object File" Block 196
5.1.2.2.6 The "Read Schedule File" Block 198
5.1.2.2.7 The "Q-80 Simulation" Block 199
5.1.2.2.7.1 Declaration Variables and Tables 199
5.1.2.2.7.2 Procedures MEM and MEMW 202
5.1.2.2.7.3 Subroutine PRINT 203
5.1.2.2.7.4 Subroutine SCHEDULE 203
5.1.2.2.7.5 Start-up of Simulation 204
5.1.2.2.7.6 Simulation of States and State Transitions 204
5.1.2.2.7.6.1 Simulation of State RESET 204
5.1.2.2.7.6.2 Simulation of State IFCH1 205
5.1.2.2.7.6.3 Simulation of State IFCH2 205
5.1.2.2.7.6.4 Simulation of the State IFCH3 205
5.1.2.2.7.6.5 Simulation of the Pseudo-state COND 205
5.1.2.2.7.6.6 Simulation of State OFCH1 206
5.1.2.2.7.6.7 Simulation of State OFCH2 206
5.1.2.2.7.6.8 Simulation of State POP1 206
5.1.2.2.7.6.9 Simulation of State POP2 206
5.1.2.2.7.6.10 Simulation of State INPUT 207
5.1.2.2.7.6.11 Simulation of the Pseudo-state EXEC 207
5.1.2.2.7.6.12 Simulation of State STORE1 207

5.1.2.2.7.6.13 Simulation of State STORE2 208
5.1.2.2.7.6.14 Simulation of State PUSH1 208
5.1.2.2.7.6.15 Simulation of State PUSH2 208
5.1.2.2.7.6.16 Simulation of State OUTPUT 208
5.1.2.2.7.6.17 Simulation of State HALTE 209
5.1.2.2.7.6.18 Simulation of State HOLD 209
5.1.2.2.7.6.19 Simulation of State HHOLD 209
5.1.2.2.7.6.20 Simulation of State HALTI 209
5.1.2.2.7.6.21 Simulation of State HALTE 209
5.1.2.2.7.6.22 Simulation of State IHHOLD 210
5.1.2.2.7.8.23 Simulation of State IEHHOLD 210
5.1.2.2.7.6.24 Simulation of State INTJAM1 210
5.1.2.2.7.6.25 Simulation of State INTJAM2 210
5.1.2.2.7.6.26 Simulation of State INTJAM3 210
5.1.2.2.7.6.27 Simulation of State Transitions 211
6. Conclusions 293
7. References 296

List of Illustrations Volume II

5.1-1 Assignment of Control Bits in the Q-80 Microword 4
5.1-2 Sample of Q-80 Microprogram Assembler Listing 5
5.1-3 GPU Equivalent Block Diagram 9
5.1-4 Q-80 Emulator Block Diagram 10
5.1-5 Flowchart of Prologue Activities Block 32
5.1-6 Flowchart of Subroutine Newword 37
5.1-7 Flowchart of Subroutine EXTRACT LABEL 39
5.1-8 Top Level Flow Chart of PASS2 47
5.1-9 Flowchart of Subroutine NEWWORD 52
5.1-10 Flowchart of Subroutine NEWPHRASE 54
5.1-11 Microword print Format for the Q-80 Microporgram Assembler 57
5.1-12 Delayed resolution of S_1 and of the A-bits. 82
5.1-13 Q80 Simulator State Diagram 175
5.1-14 Binary File Record Format 176
5.1-15 Format of Blocks in Object Code File 197
5.1-16 Organization of Arrays V and OLDV 201
5.1-17 Bit Assignment in an Entry of Table DEFTBL 202

List of Tables Volume II

5.1-1	Q80 Microprogram Assembler Accepted Phrases	8
5.1-2	Control Bit Settings	15
5.1-3	I/O Phrases	19
5.1-4	GUA Control Phrases	20
5.1-5	COUT-Phrases	21
5.1-6	MFLAGS-Phrases	22
5.1-7	The Case TYPE=6	72
5.1-8	Next State at end of Execution for HLT-instructions	182
5.1-9	State Transition at end of Instruction Execution	182

5. Support Software Design Description

5.1 Overview

The Q-80 Support Software is a package of 3 related, independently running programs which aid in the development and checkout of the Q-80. The three programs are:

- The Q-80 Microprogram Assembler
- The 8080/8085 Macro Assembler
- The Q-80 Simulator

The Q-80 Microprogram Assembler translates microprograms specified in a register transfer language into the bit patterns for the Q-80 Microprogram Control Memory. The 8080/8085 Macro Assembler is an INTEL product designed to be the general purpose, commercially available assembler for Intel's 8080/8085 product line. Questron has adapted this assembler to be used as the assembler for the Q-80. (This adaptation involved no changes to the assembler itself.)

The Q-80 Simulator simulates the Q-80 on the register and interface level. Based on object code produced by the 8080/8085 Macro Assembler it produces a diskette file and a listing which represent a complete trace of the expected behavior of the Q-80 when executing the same object code. The diskette file can be used in automated checkout of the Q-80 by means of an MDS800/Q-80 interface.

5.1.1 Q-80 Microprogram Assembler

5.1.1.1 User's Reference

Function

The Q-80 Microprogram Assembler translates symbolic microcode into binary microcode. The symbolic language accepted by the Q-80 Microprogram Assembler is basically a register transfer language. Each microword is specified as a series of register transfers, called phrases.

The process of translating these phrases into binary microcode is complicated by two factors: 1) in many cases there is more than one control configuration that will cause the execution of a given phrase, and 2) some phrases are incompatible with others when specified in the same microword. Thus it could happen that two phrases in a microword are apparently incompatible. The Q-80 Microprogram Assembler attempts to resolve such cases and will flag an incompatibility only if all possible alternate translations have also resulted in incompatibility in the given context.

In those cases where more than one translation is valid the user has the option to influence the assembler's choice by specifying some control fields explicitly. This can be regarded as a lower level feature of the language.

The Q-80 Microprogram Assembler supports conditional and unconditional branching. Symbolic labels are used.

Input

The symbolic microcode is read from an ASCII disk file named :F1:name.MIC, where "name" is an alphanumeric string of 1 to 6 characters beginning with a letter. The format of the file is compatible with the output format of the ISIS-II Text Editor.

Date File

A file called :F0:DATE is read by Q-80 Microprogram Assembler. The first 11 bytes in this file are assumed to be the current date. The file must be edited daily by the user. The format of the date is shown in the following example: 12 FEB 1977.

Output

The binary microcode is wrtten into a disk file named :F1:name.BIN where "name" is the same as the "name" in the symbolic input file specified. For example if the input file is :F1:TEST2.MIC then the binary output is placed into :F1:TEST2.BIN. If a file by that name already exists the old contents will be lost. Each microword is represented by 8 bytes. Bit assignment is as shown in Figure 5.1-1.

Listing

Figure 5.1-2 shows a short sample of Q-80 Microprogram Assembler listing. The symbolic microcode source file name and the date appear under the page header (1st page only). After the symbol table listing each symbolic microword is listed followed by values of the various control bits obtained as a result of the translation. The 12 digit hexadecimal number represents the last 6 bytes of the binary microword. The next available address used and the number of errors encountered are printed at the end of the listing.

Calling

To call the Q-80 Microprogram Assembler from ISIS-II (which prompts with a-) enter:

MICROX:F1:name.MIC

on the keyboard.

Symbolic microprogram structure

The symbolic microprogram is considered to start at the beginning of the file and ends with the line:

[END]

All blanks, tab characters (control I) and blank lines are ignored and can therefore be freely used to format the text. Comments begin with an asterisk and end at the end of the line. Everything following a semicolon up to the end of the line is also considered a comment.

R3	R2	R1	R0	RS	T2	T1	T0
DOE	M2	M1	M0	D2	D1	C1	C0
A1	A0	S1		IM	CO	F1	F0
SCY1	SCY0	CI1	CI0	I/O3	I/O2	I/O1	I/O0
GUA1	GUA0	E	FS3	FS2	FS1	FS0	ADDR9
ADDR8	ADDR7	ADDR6	ADDR5	ADDR4	ADDR3	ADDR2	ADDR1

Figure 5.1-1 Assignment of Control Bits in the Q-80 Microword.

```
                    QUESTRON
Q-80 MICROPROGRAM ASSEMBLER.                                    PAGE    1
-------------------------------------------------------------------------

      :F1:Q80.MIC                       6 JAN 1978
-------------------------------------------------------------------------

                ***************************
                *  REGISTER UTILIZATION   *
                ***************************

*               REG(0): B-REGISTER
*               REG(1): C-REGISTER
*               REG(2): D-REGISTER
*               REG(3): E-REGISTER
*               REG(4): H-REGISTER
*               REG(5): L-REGISTER
*               REG(6): BYTE-2 OF INSTRUCTION
*               REG(7): A-REGISTER
*               REG(8): MASK USED IN RST-INSTRUCTION (00111000)
*               REG(9): NO SPECIFIC ASSIGNMENT
*               REG(A): OPCODE USED IN RST INSTRUCTION
*               REG(B): BYTE-3 OF INSTRUCTION
*               REG(C): =0, CLEARED BY RESET, READ ONLY
*               REG(D): NO SPECIFIC ASSIGNMENT
*               REG(E): SP (MSB)
*               REG(F): SP (LSB)
```

```
                    QUESTRON
Q-80 MICROPROGRAM ASSEMBLER.                                    PAGE    2
-------------------------------------------------------------------------

00000000:       *** RESET ***

        PCH = DO, DO = ALC,
              REG(C) = ALC, ALC = P1B AND 0,
        GOTO 01010110;              * WHERE RESET CONTINUES

-----------                                                 ---------------
! ADDR = 00 !                                             ! C0D8E000030056 !
-------------------------------------------------------------------------
! R = C ! A = 2 ! M   = 5 !! RS  = 0 ! MIC  = 0 ! I/O = 0 ! CI  = 0 ! ADDR = 56 !
-------------------------------------------------------------------------
! T = 0 ! D = 7 ! DOE = 1 !! TS  = 0 ! PC   = 3 ! SCY = 0 ! F   = 0 ! FLAG =  0 !
-------------------------------------------------------------------------
! S = 0 ! C = 0 !         ! DIS = 0 ! PCHL = 0 !                  ! LINK =  0 !
-----------------         -------------------                     ---------------
```


Figure 5.1-2 Sample of Q-80 Microprogram Assembler Listing

The symbolic microprogram consists of a series of microword specifications. Microword specifications are separated by semicolons, i.e. each ends with a semicolon.

Each microword specification begins with a label (optional) and consists of a series of phrases. Each phrase ends in a comma, except the last in the microword which ends in a semicolon. Thus the general outline of a symbolic microprogram is:

```
label: phrase, phrase, . . . . . , phrase;
label: phrase, phrase, . . . . . , phrase;
. . . . . .
. . . . . .
label: phrase, phrase, . . . . . , phrase;

[END]
```

Labels

Microwords can be labeled in one of three ways.

1) No label. The microword is never branched to explicity. It can only be reached by executing the microword preceding it in the logical address space, which must not contain an explicit GOTO-phrase.

2) Simple label. A label of 1-8 characters (first character must be alphabetic, remaining characters may be alphanumeric) precedes the microword. The label is followed by a colon. This microword may be branched to explicitly and unconditionally from anywhere in the microprogram.

3) Label and suffix. Such a label consists of an 1-8 character label (as described above) followed by 1, 3, 4, or 7 zeros and ones enclosed between parantheses. A colon follows the closing parenthesis. Such a microword can be branched to conditionally or unconditionally from anywhere in the microprogram. Microwords with identical labels but different suffixes must be grouped together with suffixes in ascending order.

The following are some examples of labels.

```
MPY3: . . . . . .
ABCDEFGH: . . . . . .
JMP(010):
LONGLABL(1101011):
```

Phrases

Table 5.1-1 enumerates all the phrases recognized by the Q-80 Microprogram Assembler. They fall into 18 categories as indicated in the table. No more than one phrase from each category may appear in a microword. If a microword does not contain any phrase from a certain category then the default is assumed, unless the default is incompatible with another phrase. It is advisable to specify a phrase from each category unless one truly does not care which phrase is selected by the assembler, because incompatibilities with defaults occur frequently.

Semantics

Figure 5.1-3 shows a functional block diagram of the GPU data paths. This block diagram is not an accurate reflection of the physical GPU data paths but it has been verified to be functionally equivalent to the GPU in the sense that given an orginal state of the storage elements and any arbitrary sequence of microcommands that are applied to the GPU input pins, the block diagram and the real GPU will yield the same state in all storage elements and on all output pins after each microcommand. Figure 5.1-4 shows a functional block diagram of the Q-80 emulator.

The Q-80 Microprogram Assembler language is designed on the basis of these block diagrams. Each microword represents a set of "register" transfers scheduled to occur during one cycle. A cycle consists of the application of a set of control signals to the Q-80 emulator, and the subsequent issuing of a load clock pulse.

Table 5.1-1

Q-80 MICROPROGRAM ASSEMBLER
ACCEPTED PHRASES

1. *GOTO Phrases:*

```
GOTO <000+●>
GOTO <256+●>
GOTO <512+●>
GOTO <768+●>
GOTO label
GOTO label (suffix)
```

label = 1 to 8 alphanumeric characters starting with an alphabetic character.
suffix = 1, 3, 4 or 7 binary digits, or one of: S, Z, P, HHI, PHHI or EOPCODE.

2. *Register Assignment Phrases:*

```
REG(n) = ALC
REG(#) = ALC
REG(n) = SHIFTER
REG(#) = SHIFTER
```

n = a hexadecimal digit

3. *P1B Phrases:*

```
P1B = REG(n)
P1B = REG(#)
P1B = DI
```

n = a hexadecimal digit

4. *P2B Phrases:*

```
P2B = REG(n)
P2B = REG(#)
P2B = P2B
P2B = DI
```

n = an octal digit

5. *DO Phrases:*

```
DO = ALC
DO = P1B
DO = P2B
DO = OFF
```

6. *ALC Phrases:*

```
ALC = left + right
ALC = left + right + carry
ALC = left OR right
ALC = left AND right
```

Valid left-right pairs

left	right
P1B	P2B
P1B	.NOT.P2B
P1B	0
.NOT.P1B	0

carry = CO, COUT, or .NOT.COUT

7. *SHIFTER Phrases:*

```
SHIFTER = ●
SHIFTER = ALC(6–0)\\0
SHIFTER = ALC(6–0)\\1
SHIFTER = ALC(6–0)\\MXL0IN
SHIFTER =        0\\ALC(7–1)
SHIFTER =        1\\ALC(7–1)
SHIFTER = MXH0IN\\ALC(7–1)
SHIFTER =              00\\ALC(7–2)
SHIFTER =              11\\ALC(7–2)
SHIFTER = MXH1IN\\MXH0IN\\ALC(7–2)
```

8. *MX Output Phrases:*

```
MXL0OUT = ALC(0)
MXL1OUT = ALC(1)
MXH0OUT = ALC(7)
MXH1OUT = OFL
```

9. *I/O Phrases:*

```
I/O = .NOT.MEMR
I/O = .NOT.MEMW
I/O = .NOT.IOR
I/O = .NOT.IOW
I/O = HLDA
I/O = WAIT
I/O = INTES
I/O = .NOT.INTA.NOT.IFCH
I/O = HLDA.WAIT
I/O = INTER
I/O = .NOT.MEMR.NOT.IFCH
I/O = .NOT.MEMR.NOT.STACK
I/O = .NOT.MEMW.NOT.STACK
I/O = EI
I/O = DI
```

10. *GUA Control Phrases:*

```
GUA = ENABLE(CE)
GUA = ENABLE(LE1)
GUA = ENABLE(LE2)
GUA = ENABLE(LE3)
```

11. *COUT Control Phrases:*

```
COUT = GPUCOUT
COUT = DO(7)
COUT = DO(0)
COUT = COUT
```

12. *CO Phrases:*

```
CO = GPUCOUT
CO = L2
```

13. *MFLAGS Phrase:*

```
MFLAGS = ENABLE(OUT)
MFLAGS = ENABLE(SZP)
MFLAGS = ENABLE(ALL)
MFLAGS = DISABLE
```

14. *NOLOAD Phrase:*

```
NOLOAD
```

15. *E Phrases:*

```
E = 0
E = 1
```

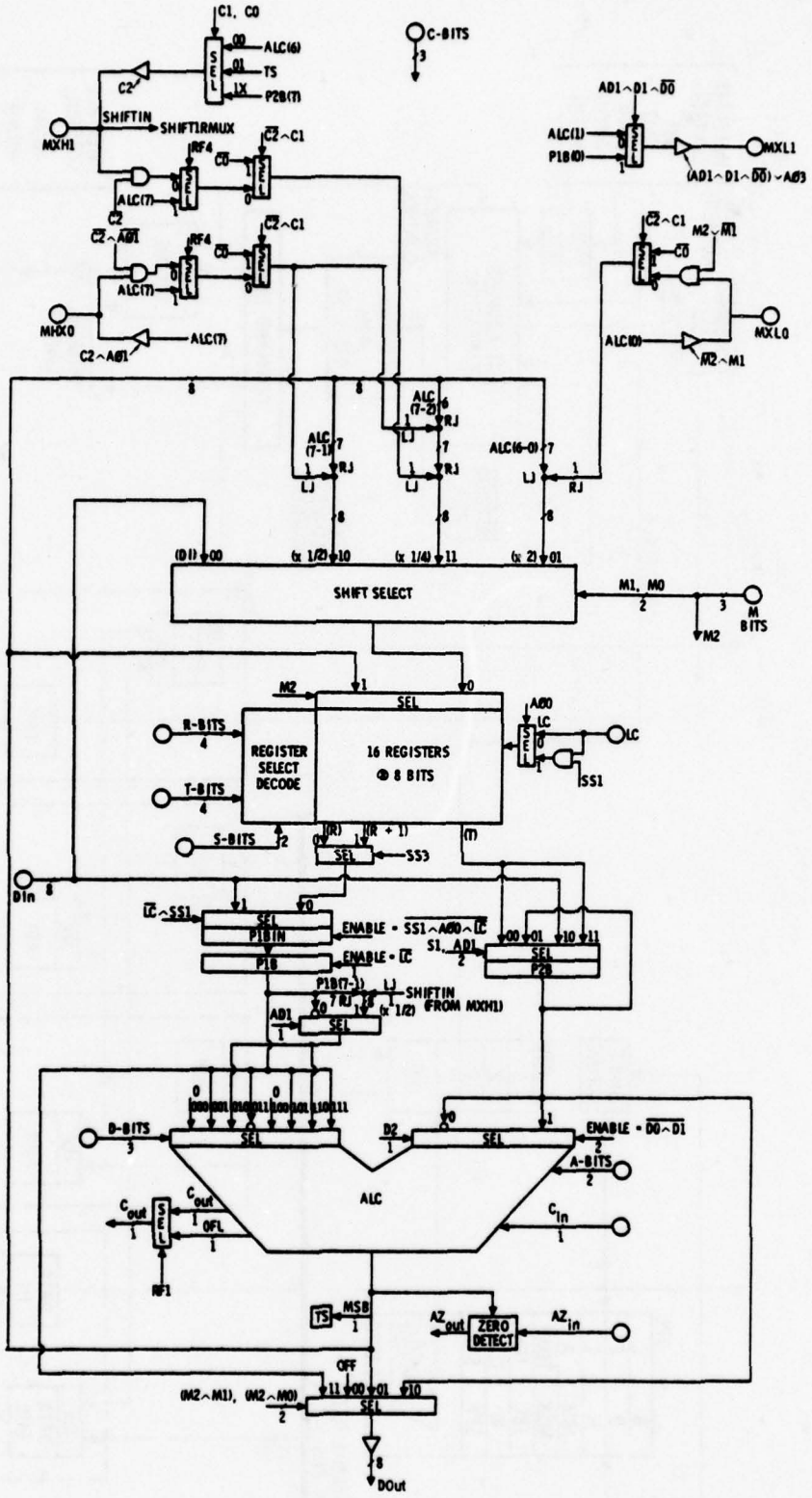


Figure 5.1-3 GPU Equivalent Block Diagram

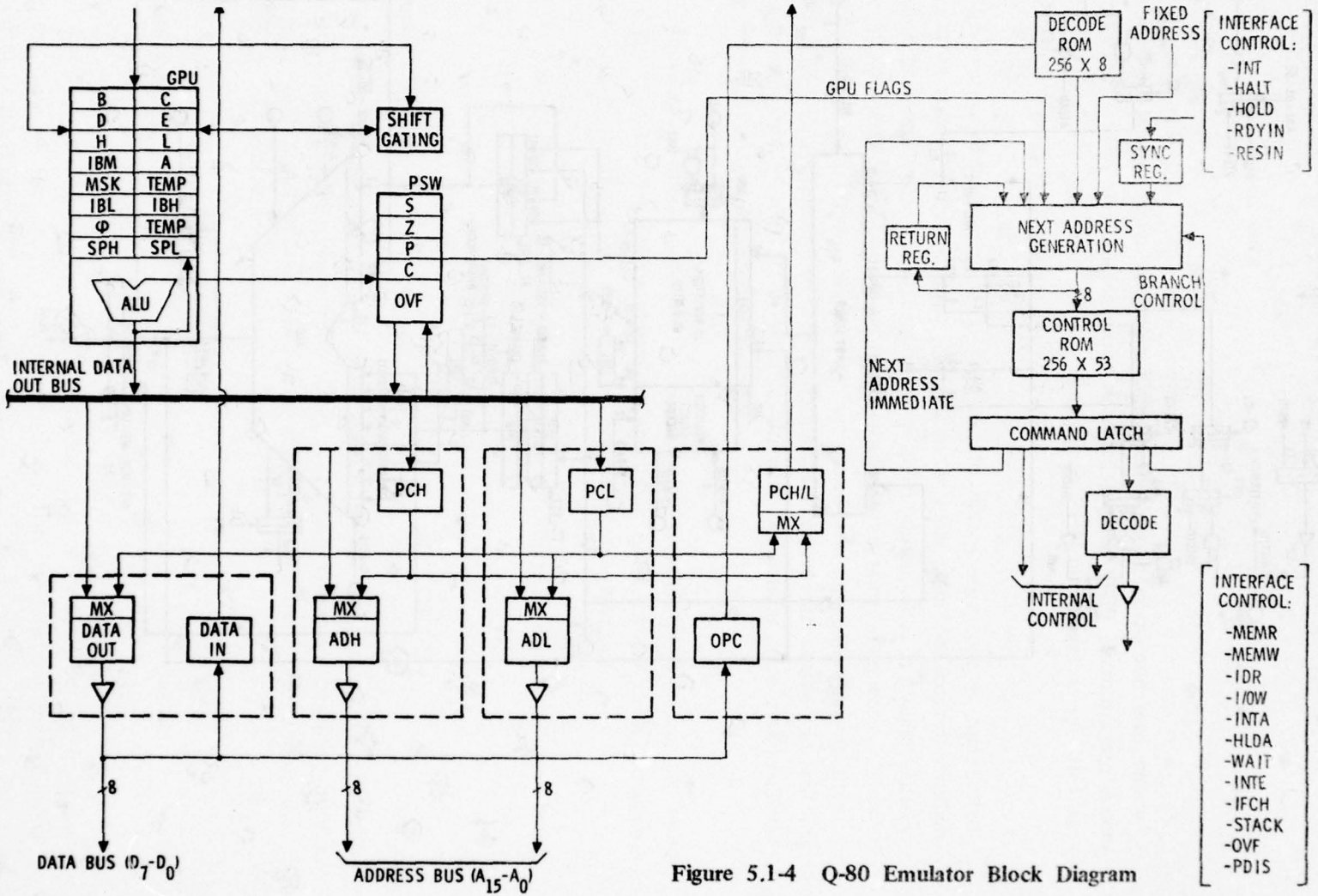


Figure 5.1-4 Q-80 Emulator Block Diagram

Each phrase specifies a register transfer. The "registers" in these register transfers are true storage elements inside or outside the GPU (e.g. P2B), GPU inputs (e.g.DI), GPU outputs (e.g. DO), pseudo register (e.g. ALC) or control signals to be stored in the microprogram control memory (e.g. CO). Two-way pins, namely the MX bits are treated as two registers, one for input and one for output (e.g. MXH1IN and MXH1OUT).

GOTO-phrases

- Absolute GOTO-phrases
  - GOTO <000 + @>
  - GOTO <256 + @>
  - GOTO <512 + @>
  - GOTO <768 + @>

These phrases cause the control bit settings:

FS = 1111

and

$ADDR_9 = 0, ADDR_8 = 0$ for 000 + @
$ADDR_9 = 0, ADDR_8 = 1$ for 216 + @
$ADDR_9 = 1, ADDR_8 = 0$ for 512 + @
$ADDR_9 = 1, ADDR_8 = 1$ for 768 + @

- Unconditional GOTO-phrases
  - GOTO *label*, where *label* consists of 1-8 alphanumeric characters, first character alphabetic

This phrase causes the following control bit settings:

ADDR = 9 MSB of *address*

$$FS = \begin{cases} 0000 \text{ if LSB of } address \text{ is } 0, \\ 0001 \text{ if LSB of } address \text{ is } 1, \end{cases}$$ where *address* is the value assigned to *label*.

- GOTO *label (suffix),* where *label* is as described above, and *suffix* consists of 1,3,4 or 7 zeros and ones.

This phrase causes the following control bit settings:

ADDR = 9 MSB of *address*

$$FS = \begin{cases} 0000 \text{ if LSB of address is } 0 \\ 0001 \text{ if LSB of address is } 1 \end{cases}$$

where *address* is the value assigned to label ORed with *suffix.*

- Conditional GOTO-phrases
  - GOTO *label (flags),* where *label* is as described above and *flags* is one of S, Z, P, HHI, RHHI, EOPCODE.

This phrase causes the following control bit settings:

ADDR = 9 MSB of *address*

$$FS = \begin{cases} 0010 \text{ if } \textit{flags} \text{ is S} \\ 0011 \text{ if } \textit{flags} \text{ is Z} \\ 0100 \text{ if } \textit{flags} \text{ is P} \\ 1000 \text{ if } \textit{flags} \text{ is RHHI} \\ 1001 \text{ if } \textit{flags} \text{ is HHI} \\ 1100 \text{ if } \textit{flags} \text{ is EOPCODE} \end{cases}$$

where *address* is the value assigned to *label.*

REG - phrases

- REG(n) = ALC, where n is a hexadecimal digit or #. This phrase causes the control bit settings:

$$\text{if n is a hexadecimal digit} \begin{cases} R = n \\ M_2 = 1 \\ RS = 0 \end{cases} \qquad \text{if n is \#} \begin{cases} R = 0 \\ M_2 = 1 \\ RS = 1 \end{cases}$$

- REG(n) = SHIFTER, where n is a hexadecimal digit or #. This phrase causes the control bit settings:

$$\text{if n is a hexadecimal digit} \begin{cases} R = n \\ M_2 = 0 \\ RS = 0 \end{cases} \qquad \text{if n is \#} \begin{cases} R = 0 \\ M_2 = 0 \\ RS = 1 \end{cases}$$

P1B - phrases

- P1B = REG(n) where n is a hexadecimal digit or #. This phrase causes the control bit settings:

$$\text{if n is a hexadecimal digit} \begin{cases} R = n \\ S_1 = 1 \\ RS = 0 \end{cases} \qquad \text{if n is \#} \begin{cases} R = 0 \\ S_1 = 1 \\ RS = 1 \end{cases}$$

- **P1B = @**

  This phrase causes the control bit settings $S_1 = 0$ and IM = 1. In addition it insures that $M \neq 000$.

- **P1B = DI**

  This phrase causes the control bit settings $S_1 = 0$ and IM = 0. In addition it insures that $M \neq 000$.

P2B = phrases

- **P2B = P2B**

  This phrase causes the control bit setting:

  $S_1 = 0$

  A = 01

- **P2B = REG(n)** where $0 \leq n \leq 7$ or n is #. This phrase causes one of the following control bit settings, depending on the context: If n is 0-7:

$$\begin{cases} T = n \\ S_1 = 0 \\ A = 00 \\ RS = 0 \end{cases} \text{ or } \begin{cases} T = n \\ S_1 = 1 \\ A = 01 \\ RS = 0 \end{cases} \text{ or } \begin{cases} T = n \\ S_1 = 0 \\ A = 10 \\ RS = 0 \end{cases} \text{ or } \begin{cases} T = n \\ S_1 = 0 \\ A = 11 \\ RS = 0 \end{cases}$$

If n is #.

$$\begin{cases} T = 0 \\ S_1 = 0 \\ A = 00 \\ RS = 1 \end{cases} \text{ or } \begin{cases} T = 0 \\ S_1 = 1 \\ A = 01 \\ RS = 1 \end{cases} \text{ or } \begin{cases} T = 0 \\ S_1 = 0 \\ A = 10 \\ RS = 1 \end{cases} \text{ or } \begin{cases} T = 0 \\ S_1 = 0 \\ A = 11 \\ RS = 1 \end{cases}$$

- P2B = DI
  This phrase causes one of the following control bit settings, depending on the context:

$$\begin{cases} S_1 = 1 \\ A = 00 \end{cases} \text{ or } \begin{cases} S_1 = 1 \\ A = 10 \end{cases} \text{ or } \begin{cases} S = 1 \\ A = 11 \end{cases}$$

DO - phrases

- DO = OFF
  This phrase causes the control bit setting DOE=0.

- DO = P1B
  This phrase causes the control bit settings: M=111, DOE = 1

- DO = P2B
  This phrase causes the control bit settings: M=110, DOE = 1

- DO = ALC
  This phrase causes the control bit settings: DOE = 1 and insures that $M \neq 11x$.

ALC – phrases

All ALC – phrases set the D-bits according to the 2 main operands *left* and *right* as shown in Table 5.1-2.

Table 5.1-2 Control bit settings based on the two main operands of an ALC-phrase.

left	*right*	*D**
P1B	.NOT.P2B	001
.NOT.P1B	0	011
P1B	P2B	101
P1B	0	111

■ ALC = *left* + *right*

This phrase causes the control bit settings:

$$\begin{cases} A_1 = 0 \\ CI = 00 \end{cases}$$

in addition to those specified in Table 5.1-2.

■ ALC = *left* + *right* + CO

This phrase causes the control bit settings:

$$\begin{cases} A_1 = 0 \\ CI = 01 \end{cases}$$

in addition to those specified in Table 5.1-2.

*** Only the 2 MSB of D are stored in the microprogram control memory. D_0 is always 1.**

- ALC = *left* + *right* + COUT

This phrase causes the control bit settings:

$$\begin{cases} A_1 = 0 \\ CI = 10 \end{cases}$$

in addition to those specified in Table 5.1-2.

- ALC = *left* + *right* + .NOT.COUT

This phrase causes the control bit settings:

$$\begin{cases} A_1 = 0 \\ CI = 11 \end{cases}$$

in addition to those specified in Table 5.1-2.

- ALC = *left* – P2B (equivalent to ALC = *left* + .NOT.P2B + CO)

This phrase causes the control bit settings:

$$\begin{cases} A_1 = 0 \\ CI = 01 \\ D = 001 \end{cases}$$

- ALC = *left* OR *right*

This phrase causes the control bit setting: A = 11 in addition to those specified in Table 5.1-2.

- ALC = *left* AND *right*

This phrase causes the control bit setting: A = 10 in addition to those specified in Table 5.1-2.

SHIFTER – phrase

- SHIFTER = @

This phrase causes the control bit settings:

$$\begin{cases} S = 01 \\ M_1 = 0 \\ M_0 = 0 \\ IM = 1 \end{cases}$$

- SHIFTER = DI

This phrase causes the control bit settings:

$$\begin{cases} S = 01 \\ M_1 = 0 \\ M_0 = 0 \\ IM = 0 \end{cases}$$

- SHIFTER = ALC(6-0)\\$\left\{\begin{array}{c} 0 \\ 1 \\ MXL0IN \end{array}\right\}$

These phrases cause the control bit settings:

$$\begin{cases} M_1 = 0 \\ M_0 = 1 \end{cases}$$

and

C = 011* for \\ 0

C = 010* for \\ 1.

The C-bits are not set for \\MXL0IN. C $\neq$ 01x* is guaranteed (since there are no other phrases that set C = 01x).

- SHIFTER = $\left\{\begin{array}{c} 0 \\ 1 \\ MXH0IN \end{array}\right\}$ \\ALC(7-1).

These phrases cause the control bit settings:

$$\begin{cases} M_1 = 1 \\ M_0 = 0 \end{cases}$$

and

C = 011* for 0\\

C = 010* for 1\\

The C-bits are not set for MXH0IN\\. C $\neq$ 01x* is guaranteed (since there are no other phrases that set C to those values).

*** Only the two LSB of C are stored in the microprogram control memory. C_2 is always 0.**

- SHIFTER = { 00 / 11 / MXH1IN\MXH0IN } \ALC(7-2).

These phrases cause the control bit settings:

$M_1 = 1$
$M_0 = 1$

and

C = 011* for 00\\
C = 010* for 11\\

The C-bits are not set for MXH1IN\\MXH0IN\\. C $\neq$ 01x* is guaranteed (since there are no other phrases that say C to those values).

MXL0OUT - Phrases

- MXL0OUT = ALC(0)

This phrase causes the control bit setting:

$M_2 = 0$
$M_1 = 1$.

MXL1OUT = Phrases

- MXL1OUT = ALC (1)

This phrase causes the control bit setting M = 011.

*** Only the two LSB of C are stored in the microprogram control memory. C_2 is always 0.**

I/O – phrases

Table 5.1-3 shows the effect of the 15 possible I/O phrases. If none is present the I/O field in the microword is set to 0000.

Table 5.1-3 I/O Phrases

Phrase	I/O Field Value
I/O = .NOT.MEMR	0001
I/O = .NOT.MEMW	0010
I/O = .NOT.IOR	0011
I/O = .NOT.IOW	0100
I/O = HLDA	0101
I/O = WAIT	01 10
I/O = INTES	0111
I/O = .NOT.INTA.NOT.IFCH	1000
I/O = HLDA.WAIT	1001
I/O = INTER	1010
I/O = .NOT.MEMR.NOT.IFCH	1011
I/O = .NOT.MEMR.NOT.STACK	1100
I/O = .NOT.MEMW.NOT.STACK	1101
I/O = EI	1110
I/O = DI	1111

GUA – control phrases

Table 5.1-4 shows the effect of the 4 possible GUA - control phrases. If none is specified GUA = ENABLE(LE3) is assumed as a default.

Table 5.1-4 GUA-control Phrases

Phrase	GUA Field Value
GUA = ENABLE(CE)	00
GUA = ENABLE(LE1)	01
GUA = ENABLE(LE2)	10
GUA = ENABLE(LE3)	11

COUT – phrases

Table 5.1-5 shows the effect of the 4 possible COUT phrases. If none is specified COUT = COUT is assumed as a default.

Table 5.1-5 COUT - Phrases

Phrase	SCY - Field Value
COUT = COUT	00
COUT = GPUCOUT	01
COUT = DO(7)	10
COUT = DO(0)	11

CO – Phrases

- CO = GPUCOUT

This phrase causes the control bit setting

- CO = 0

and is the default when no CO-phrase is specified.

- CO = L2

This phrase causes the control bit setting:

- CO = 1

MFLAGS – phrases

Table 5.1-6 shows the effect of the 4 possible macro flags control phrases. If none is specified MFLAGS = DISABLE is assumed as a default.

Table 5.1-6 MFLAGS - Phrases

Phrase	F - Field Value
MFLAGS = DISABLE	00
MFLAGS = ENABLE(OUT)	01
MFLAGS = ENABLE(	10
MFLAGS = ENABLE(ALL)	11

NOLOAD – phrase

- NOLOAD

This phrase causes the control bit setting M = 100.

E – phrase

- E = x where x is 0 or 1.

This phrase causes the control bit setting:

E = 0 or E = 1 depending on which was specified.

MXH0OUT – phrase

- MXH0OUT = ALC(7)

This phrase causes the control bit setting M = 001.

MXH1OUT – phrases

- MXH1OUT = OFL

This phrase causes the control bit setting: C = 01X*.

*** Only the two LSB of C are stored in the microprogram control memory. C_2 is always 0.**

Explicit setting of control bits.

In many cases it is possible to translate a microword in more than one way. In those cases the Q-80 Microprogram Assembler will make an arbitrary choice. In order to enable the user to influence the choice, five explicit control-bit-setting phrases are provided:

A = x
C = x
D = x
M = x
S = x

where x is a hexadecimal digit. These phrases set the respective control bits overriding any control bit settings specifed in preceding phrases without producing error messages. However, subsequent phrases that are incompatible with the explicit setting will produce error messages.

5.1.1.2 Q-80 Microprogram Assembler Computer Program Documentation*

Environment

The GPU Microprogram Assembler in written in PL/M-80 and is designed to run under the ISIS-II operating system (PL/M-80 V3.0 and ISIS-II V2.2 are Intel products).

The Q-80 Microprogram Assembler consists of 3 separately compiled parts:

- The main program MICROX
- Subroutine PASS1
- Subroutine PASS2

Each of these uses in addition to the features of PL/M-80, a number of external library routines as primitives in the program. A brief functional description of these routines follows:

OPEN – This subroutine opens a file. Given a pointer to the filename and an access code (1=read, 2=write, 3=read & write) this subroutine returns an index into the Active File Table (AFT) to be used when accessing the file and a status word.

Argument 1 = pointer to location where AFT index is to returned
Argument 2 = pointer to filename
Argument 3 = access code
Argument 4 = unused, always 0
Argument 5 = pointer to location where status word is to be returned.

This subroutine is declared in lines 2-4 of the source listing.

CLOSE – This subroutine closes a file.

Argument 1 = AFT index of file
Argument 2 = pointer to location where status word is to be returned.

This subroutine is declared in lines 5-7 of the source listing.

READ – This subroutine reads from a file. Given the AFT index of a file, a pointer to the read buffer and the number of bytes to be read, this subroutine will fill the buffer, return the number of bytes actually read and the status word.

---

* Familiarity with the Intel ISIS-II operating system and PL/M-80 compiler is assumed. It is also assumed that the reader has read the previous section describing the operation and use of the Q-80 Microprogram Assembler.

Argument 1 = AFT index of file
Argument 2 = pointer to buffer
Argument 3 = number of bytes to be read
Argument 4 = pointer of location where actual number of bytes read is to be returned
Argument 5 = pointer to location where status word is to be returned.

This subroutine is declared in lines 8-10 of the source listing.

WRITE – This subroutine writes into a file. Given the AFT index of the file, a pointer to the write buffer and the number of bytes to be written, this subroutine will write the buffer contents into the file and return the status word.

Argument 1 = AFT index of file
Argument 2 = pointer to buffer
Argument 3 = number of bytes to be written
Argument 4 = pointer to location where status word is to be returned.

This subroutine is declared in lins 11-13 of the source listing.

SEEK – This subroutine allows the repositioning of the file pointer, i.e. facilitates skipping around in the file. Many modes allow a variety of ways to move the file pointer, but some modes do not work correctly (Intel will fix in next release of ISIS-II). For this reason only two modes (modes 0 and 2) are used in this program. In mode 0 SEEK returns the block number and byte number where the filepointer is pointing (1 block = 128 bytes). In mode 2 the filepointer is set based on a blocknumber – bytenumber pair supplied as arguments.

Argument 1 = AFT index of file
Argument 2 = mode (0 or 2)
Argument 3 = pointer to blocknumber
Argument 4 = pointer to bytenumber
Argument 5 = pointer to status word

This subroutine is declared in lines 14-16 of the source listings.

DELETE – This subroutine deletes a file.

Argument 1 = AFT index of file to be deleted
Argument 2 = pointer to status word.

This subroutine is declared in lines 17-19 of the source listing.

ERROR – This subroutine displays a standard error message on the CRT. It is to be used when an abnormal status word was returned by another subroutine.

Argument 1 = error number
Argument 2 = pointer to status word.

This subroutine is declared in lines 20-22 of the source listing.

EXIT – This subroutine terminates a program and returns to ISIS-II. There are no arguments. This subroutine is declared on lines 23-24 of the source listing.

ALPHA – Simplified version of WRITE. It is mostly used to display alphanumeric data on the CRT or on the printer.

Argument 1 = AFT index of file
Argument 2 = number of bytes to be written
Argument 3 = pointer to buffer.

This subroutine is declared in lines 25-27 of the source listing

BINARY – Converts 1-8 least significant bits in a given byte to ASCII representation and writes those characters into the file indicated (mostly CRT or LP).

Argument 1 = AFT index of file
Argument 2 = number of bits to be displayed
Argument 3 = data (1 byte).

This subroutine is declared in lines 28-30 of the source listing.

HEX – Converts the contents of a buffer to its hexadecimal representation in ASCII and writes those characters into the file indicated (mostly CRT or LP).

Argument 1 = AFT index of file
Argument 2 = number of hexadecimal digits to be displayed
Argument 3 = pointer to buffer.

This subroutine is declared in lines 31-33 of the source listing.

INTEG – This subroutine converts a byte (considered as an 8 bit 2's complement interger to signed decimal in ASCII and writes those characters into the file indicated (mostly CRT or LP).

Argument 1 = AFT index of file
Argument 2 = data (1 byte).

This subroutine is declared in lines 34-36 of the source listing.

INTEGD – This subroutine converts a two-byte unsigned binary number (type ADDRESS) to its 5-digit decimal equivalent in ASCII and writes those characters into the file indicated (mostly CRT or LP).

Argument 1 = AFT index of file
Argument 2 = data (2 bytes)

This subroutine is declared in lines 37-39 of the source listing.

PAGE – Sends the form-feed character (hexadecimal 0C) to the file indicated. If the file is the printer, the paper is advanced to the top of the next page. The only argument is the AFT index of the file. This subroutine is declared on lines 40-42 of the source listing.

PPAGE – Like PAGE, except the vertical-tab character (hexadecimal 0B) is sent. The Centronics 306 printer equipped with a vertical format unit (VFU) can be programmed to skip in response to this character to either the first or second top of page, such that the face of first page folds on top of the listing (not the back of the first page). This subroutine is declared in lines 43-45 of the source listing.

SKIP – Sends a carriage-return and a line-feed character (0D0A) to the file specified. The only argument is the AFT index of the file. This subroutine is declared in lines 46-48 of the source listing.

SPACE – This subroutine sends a specified number of blanks to the file indicated.

Argument 1 = AFT index of file
Argument 2 = number of blanks.

This subroutine is declared in lines 49-51 of the source listing.

EQUAL – This function compares two given bit strings and returns the value FF if they are equal, or 00 if they are not.

Argument 1 = pointer to first string
Argument 2 = pointer to second string
Argument 3 = number of bytes to be compared

This function is declared in lines 52-54 of the source listing.

SETBIT – This subroutine can set up to 7 bits in a bit string.

Argument 1	=	pointer to string
Argument 2	=	start-bit number (0-origin)
Argument 3	=	number of bits to be set
Argument 4	=	data (1 byte). The least significant bits are used to change the bit string.

This subroutine is declared in lines 55-57 of the source listing.

EQUBIT – This function compares a given bit string to a substring of a longer bit string and returns the value FF if they are equal, or 00 if they are not.

Argument 1	=	pointer to long bit string
Argument 2	=	start-bit number (0-origin)
Argument 3	=	number of bits to be compared
Argument 4	=	short bit string (1 byte, number of least significant bits to be compared is indicated by argument 3).

This subroutine is declared in lines 58-60 of the source listing.

The computer program description, which follows, describes a program with deep nesting of blocks and loops. Accordingly, the discussion begins at the highest level and proceeds to deeper levels of nesting and at the same time to levels of increasing details. To make the relationship of the various sections that follow as clear as possible, the sections are numbered like the Dewey Decimal System. Thus for a section numbered 3.1.8, further details can be found in secitons 3.1.8.1, 3.1.8.2, . . . etc., To find the relationship of section 3.1.8 to sections 3.1.1, 3.1.2, . . .etc., read section 3.1.

5.1.1.2.1 The Main Program MICROX - The outermost block of the MICROX represents the global environment. It contains:

1. declarations of external global procedures (described above, lines 2-60)
2. declarations of global variables and constants (lines 61-69)
3. declarations of internal global procedures (lines 70-96)
4. declarations of external subroutines PASS1 and PASS2 (lines 97-100)
5. The prologue activities block (lines 101-167)
6. Calls to PASS1 and PASS2 (lines 168-169).

5.1.1.2.1.1 Declaration of external global procedures

These procedures were described above in detail.

5.1.1.2.1.2 Declaration of global variables

The global variables fall into 4 categories:

- AFT indexes for the source input file, the binary output file, the line printer, the keyboard and the CRT. (SI, BO, LP, KB, CRT).
- Special character constants (CR, LF, TAB).
- Global pointers and counters, the end-of-symbol-table pointer (ENDSYMBTAB), the pointer to the first available address following the end of the microprogram (FIRSTAVAILADDR), the error count (E).
- Paging counters: line number (LINENO) and page number (PAGENO).

5.1.1.2.1.3 Declaration of internal global procedures

There are two global internal subroutines, both related to the page-formatting of the output listing: HEADER (lines 70-89) and EOL (stands for end-of-line, lines 90-96).

- Subroutine HEADER has one parameter, SW, which indicates whether this is the first page of the listing (SW=1) or not (SW=0). Depending on the value of SW, subroutine PPAGE or PAGE is called to advance to the top of a new page (lines 72-74). Then a page header is printed, as shown in Figure 9. Finally LINENO is set to 5 and PAGENO is incremented (lines 87-88).

- Subroutine EOL is called from throughout the program, whenever a line is ended. It increments LINENO (line 91), calls subroutine SKIP which writes a carriage-return and a line-feed to the printer (line 92), and if LINENO $>$ 61 it calls HEADER (lines 93-95).

5.1.1.2.1.4 Declaration of external procedures PASS 1 and PASS 2.

These procedures are declared in lines 97-100. Neither of these subroutines has any arguments.

5.1.1.2.1.5 Prologue activities block.

Figure 5.1-5 shows a flowchart of this block. The following local variables are declared in this block:

- LINE, an array of 256 bytes, used as a read buffer
- FN, an array of 15 bytes, used as a character string holding the source file name.
- LENGTH used to return the number of bytes read when calling READ.
- LINEPTR, FNPTR used as indexes into LINE and FN respectively
- I, used as a general purpose loop counter
- STATUS, ST used as status words in calls to READ, OPEN, CLOSE.
- DATEAFT, used to hold the AFT index of the date file
- DATE, an array of 11 bytes, used to hold the character string representing the date.

Execution proceeds as follows. The remainder of the command line, entered on the keyboard when MICROX was called from ISIS-II, is read into LINE (line 108). This may or may not contain the filename. If the number of characters read (LENGTH) is 2 (i.e. only a CR-LF was read) then a prompt has to be sent to the CRT (line 111) and the keyboard is read into LINE (line 112). If LENGTH=2 again, EXIT is called terminating the program (line 113). In all other cases, LINE contains the filename before execution of line 111. Lines 116-119 position LINEPTR to point to the first non-blank character in LINE. In lines 120-125 the filename is copied from LINE to FN. At most 14 characters are copied and copying stops whenever a blank or CR is encountered. Note that a legal ISIS-II filename can not be longer than 14 characters:

:F1:XXXXXX.XXX

Next blanks are inserted at the trailing end of FN (at least one blank, lines 126-128). The source file is opened in line 129. Bad filenames and non-existent files are detected, appropriate error messages are printed and EXIT is called to terminate the program (lines 130-141).

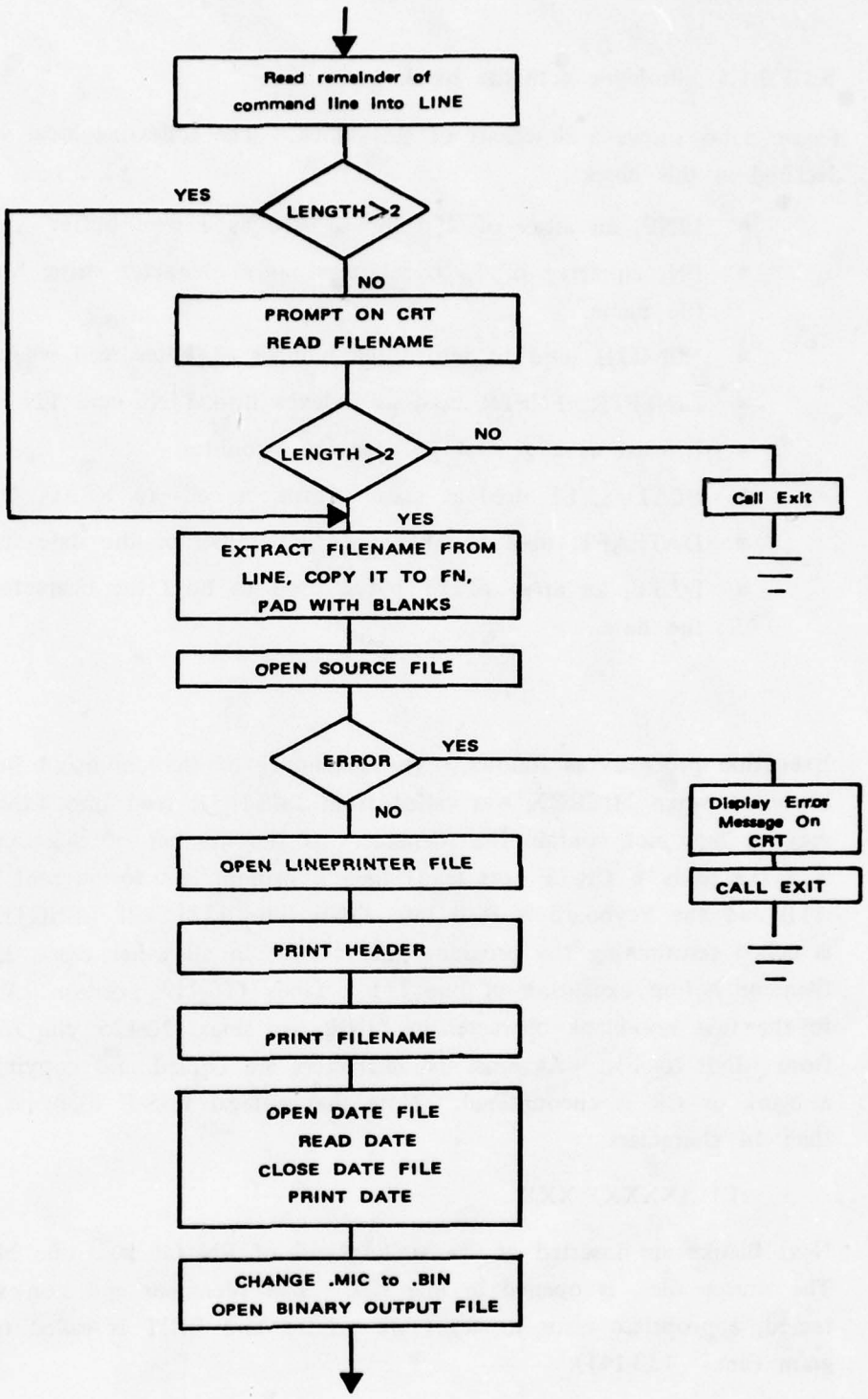


FIGURE 5.1-5 FLOWCHART OF PROLOGUE ACTIVITIES BLOCK

Next the lineprinter file is opened (line 142). PAGENO is initialized to 1 (line 143) and HEADER is called with argument=1 (causes a call to PPAGE), then the filename (FN) is printed under the page header (lines 144-147). To read the date from file :F0:DATE, the file has to be opened, read and closed (lines 148-150). Then the date is printed on the same line where the filename was printed across the page (line 154-156).

Last the file extension has to be changed from .MIC to .BIN in FN (lines 157-165) and the binary output file is opened (line 166).

Subroutines PASS1 and PASS2 are called, once each, in lines 168-169.

5.1.1.2.2 Subroutine PASS1

The outermost block, enclosing subroutine PASS1 is labelled MICRO1. This block links PASS1 to the global environment by means of declarations of EXTERNAL variables and procedures, and it contains the code for subroutine PASS1. .

The block MICRO1 contains:

1. declarations of external global procedures (lines 2-60)
2. declarations of external global variables and constants (lines 61-67)
3. declarations of global procedures defined in the main program MICROX (lines 68-72)
4. the body of subroutine PASS1 (lines 73-445)

2.1 Declarations of external global procedures

These are the same declaration found at the beginning of MICROX and are described in detail at the beginning of this chapter.

5.1.1.2.2.2 Declaration of external variables and constants

These are variables and constants defined in the main program MICROX. They fall into 4 categories:

- AFT indexes for various files (SI, BO, LP, KB, CRT) (line 61)
- Special character constants (CR, LF, TAB) (line 62)
- Global pointers and counters:
  E – the error count (line 63)
  ENDSYMBTAB – pointer to the end of the symbol table (line 64)
  FIRSTAVAILADDR – pointer to the first free microword following the end of the microprogram (line 65)
- Paging counters:
  LINENO and PAGENO (lines 66-67)

5.1.1.2.2.3 Declaration of global procedures

These are two global procedures, HEADER and EOL, which are defined in the main program MICROX (lines 68-72).

5.1.1.2.2.4 Subroutine PASS1

Subroutine PASS1 has no parameters. It has the PUBLIC attribute because the main program MICROX calls PASS1 as an external subroutine.
Subroutine PASS1 contains the following:

1. declarations of variables local to PASS1 (lines 74-83).
2. declaration of subroutines local to PASS1 (lines 84-219)
3. initializations (lines 220-224)
4. the code to build the symbol table and partially produce the binary output file (lines 225-435).

5.1.1.2.2.4.1 Declaration of variables local to PASS1

The following variables are declared local to PASS1:

- BUFF, an array of 256 bytes, used as read buffer for the symbolic source input file.
- BUFFPTR, used as index to the character string BUFF
- CODE, an array of 256 bytes, used to hold one symbolic microword after blanks, tabs, comments, etc., have been eliminated.
- CODEPTR, used as index to the character string CODE.
- LABL, an array of 8 bytes, used to hold a label.
- SUFFIX, used to hold the value of the suffix.
- SUFFIXLENGTH, used to hold the number of bits in SUFFIX that really make up the suffix for the label of the current microword.
- SYMBTABPTR, used as pointer to array MEMORY which is used as the symbol table.
- STATUS, used in calls to SEEK.

5.1.1.2.2.4.2 Declaration of subroutines local to PASS1

The following subroutines are local to PASS1.

- BUFFADV (lines 84-101),
- NEWWORD (lines 102-145),
- EXTRACTLABEL (lines 146-219).

5.1.1.2.2.4.2.1 Subroutine BUFFADV

The subroutine is used when reading text from the input buffer BUFF, one character at a time. If the buffer is empty it is refilled. The subroutine

increments BUFFPTR (line 86) and tests if BUFFPTR=256 (line 87). If it is not, control returns to the point of call. If BUFFPTR=256 then BUFF must be refilled (line 89) and BUFFPTR must be reset to 0 (line 99).

To prevent infinite loops, when symbolic source files are not properly terminated by [END], a test is made on the number of characters read (local variable NCHAR). If it is zero, as would be the case only if a source file lacked proper termination, the program terminates by calling EXIT (lines 92-98).

5.1.1.2.2.4.2.2 Subroutine NEWWORD

This subroutine sets up the environment to process the labels of a new microword. Its functions are:

- isolating the symbolic source text for one microword,
- compressing blank, tab, CR, LF characters and comments out of the symbolic code and placing it into array CODE,
- resetting CODEPTR,
- setting the flag EOF when [END] is found.

Subroutine NEWWORD contains a subroutine, CODEADV (lines 104-107), and its own body of code.

5.1.1.2.2.4.2.2.1 Subroutine CODEADV

This subroutine advances the pointer to array CODE, CODEPTR, by one (line 106). Microwords longer than 256 character are truncated. (No legal microword of more than 256 character is possible anyway).

5.1.1.2.2.4.2.2.2 The body of subroutine NEWWORD

A top level flowchart of this subroutine is shown in Figure 5.1-6.

Resetting CODE consists of filling the array with blanks (lines 108-110) and setting CODEPTR to zero (line 111).

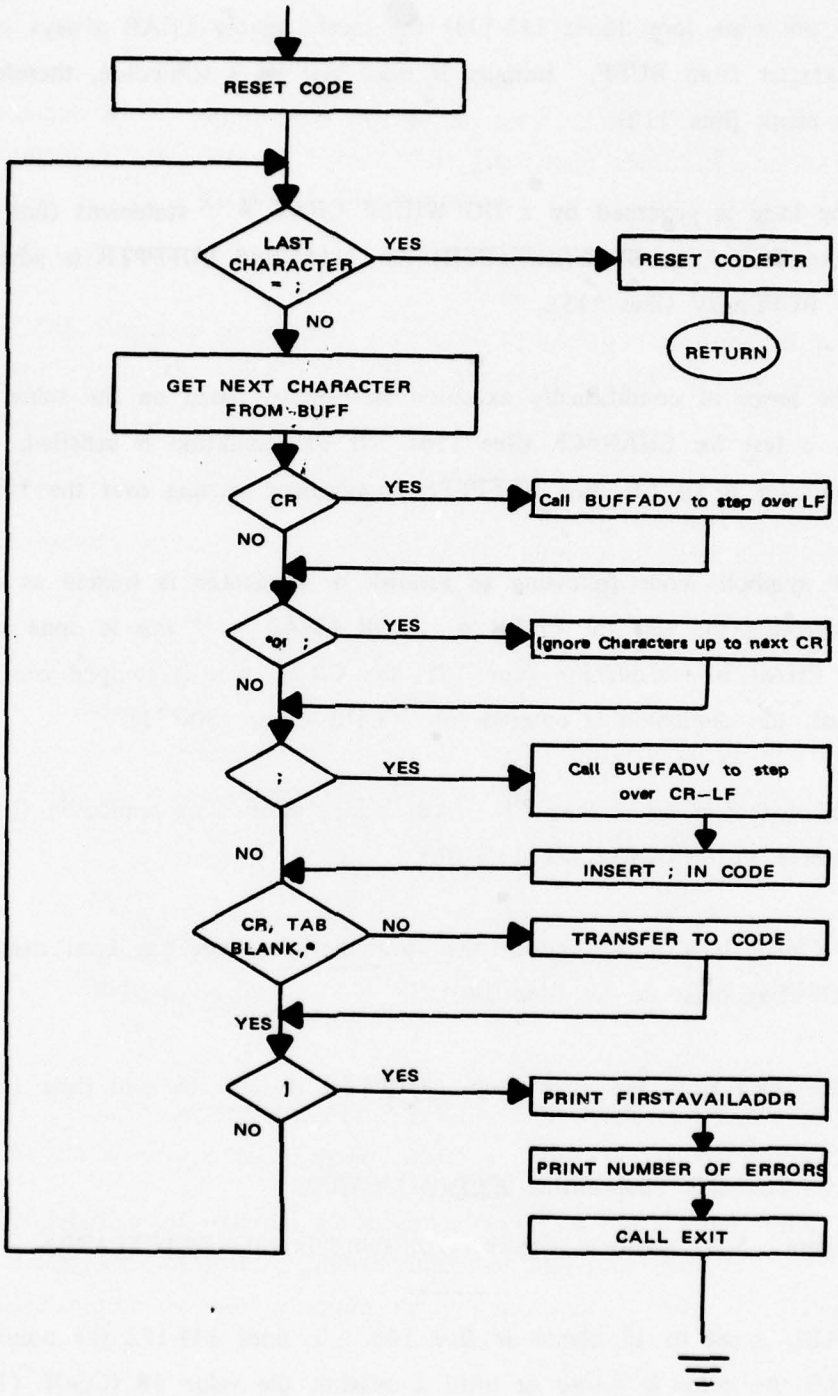


FIGURE 5.1-6 FLOWCHART OF SUBROUTINE NEWWORD

In the main loop (lines 113-143) the local variable CHAR always contains the latest character from BUFF. Initially it must not be a semicolon, therefore CHAR is set to blank (line 112).

The loop is governed by a DO WHILE CHAR ≠ ';' statement (line 113). CHAR is updated from BUFF(BUFFPTR) (line 114) and BUFFPTR is advanced by a call to BUFFADV (line 115).

The series of conditionally executed statements, based on the value of CHAR, begins by a test for CHAR=CR (line 116). If the condition is satisfied, and if the next character is an LF then BUFFPTR is advanced passing over the LF (line 119).

All symbolic code following an asterisk or semicolon is treated as a comment. Following the test for CHAR = '*' OR CHAR = ';' this is done in lines 122-124. If CHAR is a semicolon (line 125) the CR-LF pair is stepped over (lines 127-128). Last, the semicolon is entered into CODE (lines 130-131).

All characters other than CR, TAB, blank, asterisk or semicolon (line 133) are simply entered into CODE (lines 135-136).

If CHAR is a] the end of the symbolic microcode has been reached and the EOF flag must be set (line 140).

After exit from the main loop, CODEPTR is reset to zero (line 144).

5.1.1.2.2.4.2.3 Subroutine EXTRACTLABEL

Figure 5.1-7 shows a flowchart of subroutine EXTRACTLABEL.

LABL is set to all blanks in line 148. In lines 149-152 the pointer L is moved until the colon is found or until L reaches the value 18 (CODE (17) is the last place where the colon could legally be located). If L<18 the colon has been located.

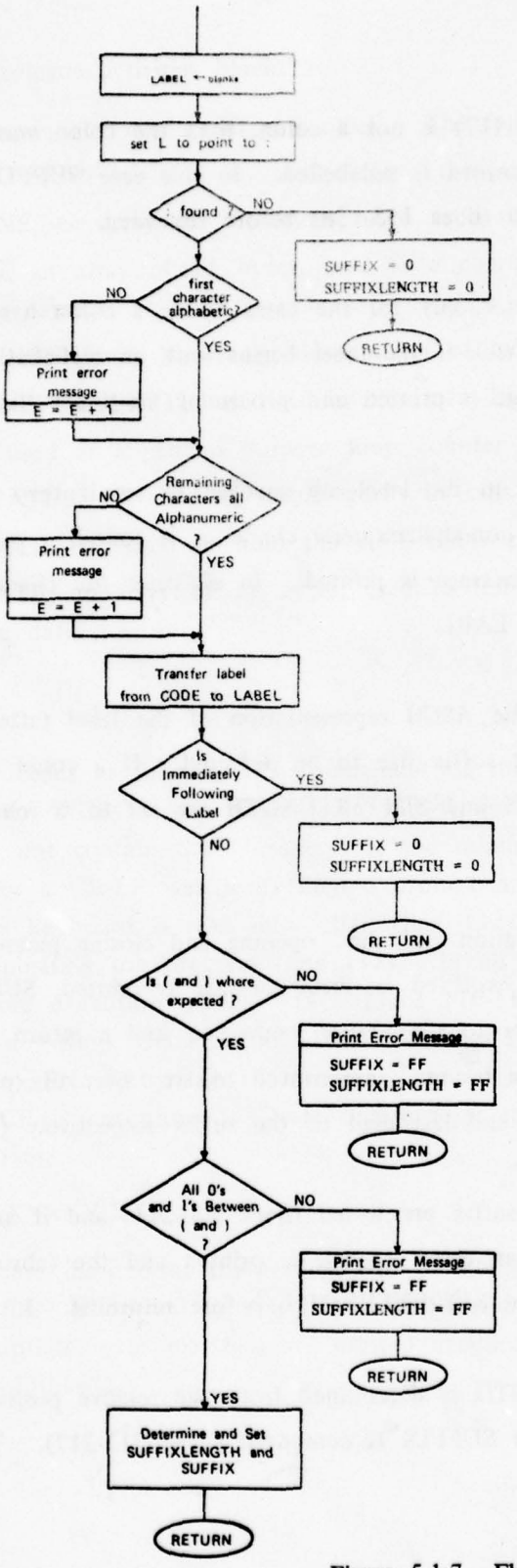


Figure 5.1-7 Flowchart of Subroutine **EXTRACT LABEL**

If L = 18 and CODE (17) is not a colon, then the colon was not found, meaning that the microword is unlabelled. In this case SUFFIX and SUFFIXLENGTH must both be set to 0 (lines 153-158) before returning.

Thus processing continues only for the cases where a colon has been located. The next task is to check if the label begins with an alphabetic character (line 159). If not, an error message is printed and processing continues (lines 160-166).

Next all the characters in the label are checked to see if they are alphanumeric (lines 167-180). If a non-alphanumeric character is found, a question mark is substituted and an error message is printed. In all cases the characters making up the label are transfered to LABL.

After LABL contains the ASCII representation of the label (after line 180), the case of a label without suffix has to be detected. If a colon follows the label (line 181) then SUFFIX and SUFFIXLENGTH are set to 0 followed by return (lines 182-186).

In line 187 correct location of the opening and closing parantheses is checked. If they are not where expected an error message is printed, SUFFIX and SUFFIXLENGTH are set to the absurd value 255 and a return is executed (lines 188-197). The pointers I and L are moved to step over the parantheses and now point to the first and last digit of the suffix respectively (lines 198-199).

Next, all digits in the suffix are tested (lines 200-211) and if any are found to be other than 0 or 1 an error message is printed and the (absurd) value 255 is assigned to SUFFIX and SUFFIXLENGTH, before returning.

Otherwise SUFFIXLENGTH is determined from the relative position of pointers I and L (line 212) and SUFFIX is computed (lines 213-217). Thereafter a return is executed (line 218).

5.1.1.2.2.4.3 Initializations

The error count E is initialized to 0, the symbol table pointer SYMBTABPTR is initialized to 0, the end-of-file flag EOF is set false (i.e. 0) and BUFFADV is called with BUFFPTR = 255, causing it to fill BUFF. (All the above in lines 220-224).

5.1.1.2.2.4.4 Code to build symbol table and partially produce binary output file

This part of PASS1 proceeds in the following major steps:

1. Build symbol table (lines 225-242)
2. Abort if any errors detected (lines 243-250)
3. Check correctness of symbol table (lines 251-295)
4. Abort if any errors detected (lines 296-304)
5. Compress symbol table (lines 305-318)
6. Assign addresses to labels (lines 319-369)
7. Allocate space and write address on disk (lines 370-386)
8. Produce symbol table listing (lines 387-441)
9. Reopen source and binary files for PASS2 (lines 442-443)

5.1.1.2.2.4.4.1 Build symbol table

The loop in lines 226-241 is executed once for each microword until the EOF flag becomes true.

Following a previous call to NEWWORD (line 225 the first time, line 240 at all other times), EXTRACTLABEL is called, which updates LABL, SUFFIX and SUFFIXLENGTH according to the label of the current microword.

The array MEMORY (which is limited only by the end of physical memory) is used to hold the symbol table. If an imminent memory overflow is detected (line 228), then the assembly is aborted after printing an error message (lines 229-235).

Otherwise LABL, SUFFIXLENGTH, and SUFFIX are copied into the symbol table (lines 236-238), and the symbol table index, SYMBTABPTR, is incremented by 12 (the length of a symbol table entry is 12) (line 239). NEWWORD is called (line 240) fetching the next microword.

After exit from the loop ENDSYMBTAB is set to point to the last byte used for the symbol table in array MEMORY (line 242).

5.1.1.2.2.4.4.2 Abort if any errors detected

If any errors have been detected in building the symbol table, the assembly is aborted at this point (lines 243-250). Error messages were printed at the time the errors were detected.

5.1.1.2.2.4.4.3 Check correctness of the symbol table

The following aspects of symbol table correctness are checked:

- suffix length is 0,1,3,4, or 7
- length of blocks agrees with suffix length
- suffix length consistent throughout each block
- suffixes in ascending order in each block

These tests will also detect absurd values of suffixes and suffixlengths.

The loop in lines 257-295 is executed once for each block of microwords and terminates when the end of the symbol table is reached.

The tests for suffixlength being 0,1,3,4, or 7 is made in line 259. If any other suffixlength is indicated in the symbol table, an error message will be printed (lines 260-266).

The length of the block is computed based on the suffixlength (lines 267-269). In lines 270-274 the true length of the block is determined by counting the

number of consecutive symbol table entries with identical label fields.

If these two values disagress (line 275) then an error message is printed (lines 276-283). If they agree, the ascending order of suffixes is checked (lines 284-293) and if any suffix is out of order an error message is printed.

5.1.1.2.2.4.4.4 Abort if any errors found

If any errors have been detected during the checking of the symbol table, the assembly is aborted (lines 296-304).

5.1.1.2.2.4.4.5 Compress symbol table

Once the block structure of the symbol table has been found to be correct, there is no further reason to retain one entry for each microword. By retaining only one entry for each block, all searches of the symbol table become much faster.

Compression of the symbol table is done by copying only the first entry from each block (lines 305-318). ENDSYMBTAB is adjusted for the shorter symbol table.

5.1.1.2.2.4.4.6 Assignment of addresses

The optimal address assignment algorithm is used, which allocates the largest blocks first. This leads to no wasted microprogram control memory space due to fragmentation.
Addresses are assigned to all blocks of 128 (lines 323-331), followed by the blocks of 16 (lines 332-340), followed by the blocks of 8 (lines 341-349), followed by the blocks of 2 (lines 350-358), followed by single microwords (lines 359-367). FIRSTAVAILADDR is set to point after the last address used (line 368).

5.1.1.2.2.4.4.7 Allocate space and write address on disk

Since the entries in the symbol table are in the order in which blocks were encountered in the source code, it is a simple matter to sequentially allocate

8 bytes for each microword on the disk and write the assigned address into the first two. Then PASS2 can just fill in the remaining 6 bytes as it translates the microwords in the order of their occurence in the source code. Lines 370-386 implement the above in a straight forward manner.

5.1.1.2.2.4.4.8 Symbol table listing

The code in lines 387-441 produces the symbol table listing. It consists of three columns, 57 lines per page (53 on the first page). For each entry the label, the suffixlength and the address are shown.

The three-column listing is implemented by means of three pointers into the symbol table (I1,I2,I3). The distance between I1 and I2 as well as between I2 and I3 is 57 symbol table entries (53 on the first page). If I3 or both I2 and I3 point past the end of the symbol table, then nothing is printed in the respective columns. When I1 points past the end of the symbol table the listing is complete.

Initially I1 is set to point to the first symbol table entry, I2 to the 54-th, I3 to the 107-th (lines 390-392) and the line counter L is set to 4 (lines 393). The first page is 4 lines shorter than any subsequent page because the header contains the file name and the date in addition to the usual header.

The loop in lines 394-440 is executed once for each line of symbol table listing printed, the loop is terminated when I1 points past the end of the symbol table. The code in lines 395-402, 404-413 and 415-424 prints the entries in column 1, 2 and 3 respectively.

When I3 or both I2 and I3 point past the end of the symbol table then column 3 or both columns 2 and 3 are left blank. This is implemented by the tests in lines 403 and 414.

At the end of each iteration, after a call to EOL (line 425), L is incremented (line 426) and tested (line 427). If L has not reached the value 57 yet, then the pointers I1, I2 and I3 are incremented by 12 (lines 434-438). If L = 52 then L is reset to 0(line 429), I1 is set to point to the symbol table entry

following the last one printed in column 3 (line 430), I2 is set to point 57 entries past I1 (line 431) and I3 is set to point 57 entries past I2 (line 432). (Thus, unlike the first page, which had 53 lines, any subsequent pages have 57 lines).

After exit from the loop HEADER is called causing a page advance unless L = 0 (lines 440-441).

5.1.1.2.2.4.4.9 Reopen source and binary files

This is accomplished by two calls to SEEK (lines 443-444).

5.1.1.2.3 Subroutine PASS2

The outermost block, enclosing subroutine PASS2 is labelled MICRO2. This This block links PASS2 to the global environment by means of declarations of EXTERNAL variable and procedures, and it contains the code for subroutine PASS2.

The block MICRO2 contains:

1. declarations of external global procedures (lines 2-60)
2. declarations of external global variables and constants (lines 61-67)
3. declarations of global procedu res defined in the main program MICROX (lines 68-72)
4. the body of subroutine PASS2 (lines 73-1898)

5.1.1.2.3.1 Declarations of external global procedures

These are the same declaration found at the beginning of MICROX and are described in detail at the beginning of this chapter.

5.1.1.2.3.2 Declaration of external variables and constants

These are variables and constants defined in the main program MICROX. They fall into 4 categories:

- AFT indexes for various files (SI, BO, LP, KB, CRT) (line 61)
- Special character constants (CR, LF, TAB) (line 62)
- Global pointers and counters:

  E - the error count (line 63)
  ENDSYMBTAB - pointer to the end of the symbol table (line 64)
  FIRSTAVAILADDR - pointer to the first free microword following the end of the microprogram (line 65)

- Paging counters: LINENO and PAGENO (lines 66-67)

5.1.1.2.3.3 Declaration of global procedures

These are two global procedures, HEADER and EOL, which are defined in the main program MICROX (lines 68-72)

5.1.1.2.3.4 Subroutine PASS2

Subroutine PASS2 has no parameters. It has the PUBLIC attribute because the main program MICROX calls PASS2 as an external procedure.

Figure 5.1-8 shows a top level flowchart of subroutine PASS2. The symbolic input file is read line by line until a line containing a semicolon is encountered (i.e. 1 microword is read). Blanks, tabs (control I), carriage returns, line feeds and comments are eliminated.

The phrase translating loop is entered which translates one phrase per iteration, until a semicolon is encountered. As each phrase is translated, wherever no alternate translations are possible, the respective control bits are set in the microword buffer. Where more than one translation is possible, one of a set of flags is set and the final resolution of the control bit setting to be chosen is delayed until all phrases of the microword have been examined.

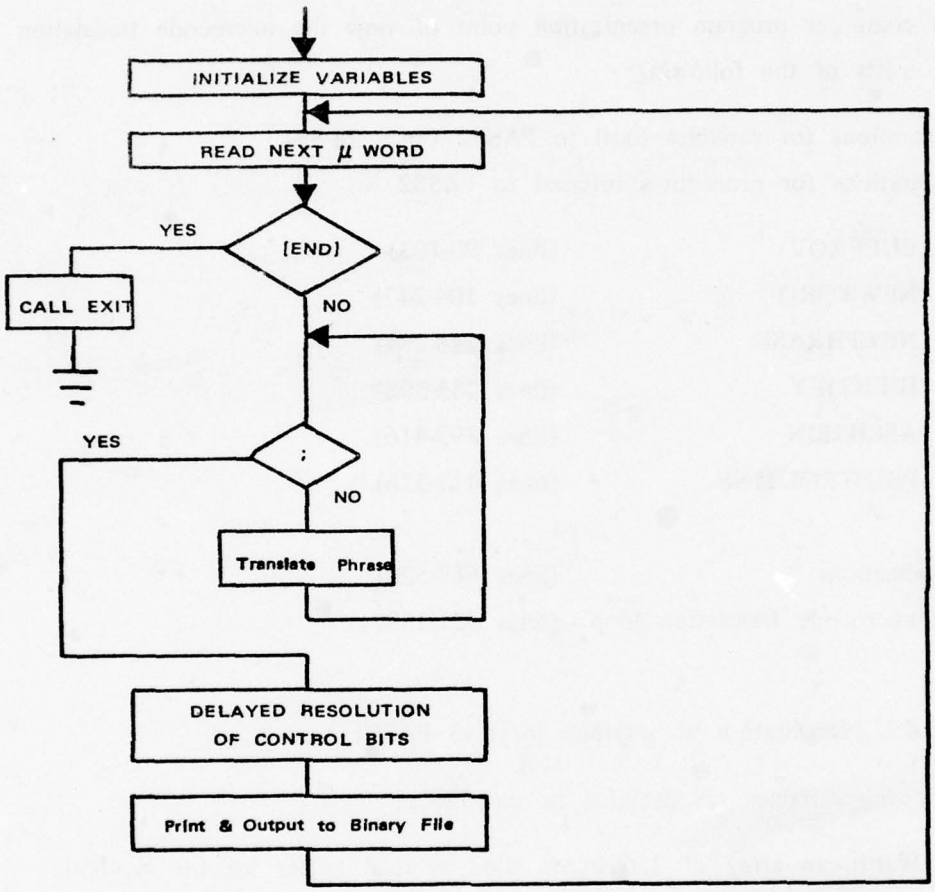


Figure 5.1-8 Top level flow chart of PASS2

Thus after the semicolon is encountered these delayed resolutions have to be made. Thereafter the control bit settings are printed on the output listing and the microword buffer is output to the binary output file.

From a computer program organization point of view the microcode translation block consists of the following:

1. declarations for variables local to PASS2 (lines 74-89)
2. declarations for procedures internal to PASS2
   - BUFFADV (lines 90-103)
   - NEWWORD (lines 104-247)
   - NEWPHRASE (lines 248-284)
   - IDENTIFY (lines 285-398)
   - ASCII2BIN (lines 399-416)
   - PRINTROUTINE (lines 417-516)
3. initializations (lines 517-520)
4. the microcode translation loop (lines 521-1897)

5.1.1.2.3.4.1 Declaration of variables local to PASS2

The following variables are declared in this block:

- BUFF, an array of 256 bytes, used as read buffer for the symbolic source input file,
- LINE, an array of 80 bytes, used as print buffer for the output listing,
- CODE, an array of 256 bytes, used to hold one symbolic microword after blanks, tabs, comments, etc. are eliminated,
- PHRASE, an array of 33 bytes, used to hold one phrase,
- ADDR, a two-byte variable, used to hold the microword address,
- BUFFPTR, CODEPTR, used to index the character strings BUFF and CODE, respectively,

- P1BDIPHRASEPRESENT used as delayed resolution flag to indicate the presence of the phrase P1B=DI,
- P2BREGPHRASEPRESENT used as delayed resolution flag to indicate the presence of the phrase P2B=REG(t),
- DOALCPHRASEPRESENT used as delayed resolution flag to indicate the presence of the phrase DO=ALC,
- NOTAD1 used as delayed resolution flag where A $\neq$ 01 is required,
- I used as a general purpose temporary variable (e.g. loop counter),
- PHVECT, an array of 18 bytes, used to flag the presence of a phrase of a certain type (see comment in source listing line 86),
- TYPE used to indicate type of phrase being processed, also index for array PHVECT,
- FIELD, an array of 22 bytes, used to indicate whether control bits in the microword buffer have been set by previous phrases or owe their value to the default (see comment in source listing line 87).
- MWORD, an array of 6 bytes, used as the microword buffer, also output buffer to binary output file,
- TBL, an array of 256 bytes, used in the delayed resolution of control bits A and S_1 (see comment in source listing line 89).

5.1.1.2.3.4.2 Procedures internal to PASS2

The 6 procedures internal to PASS2 are described individually below.

5.1.1.2.3.4.2.1 Subroutine BUFFADV

This subroutine is used when reading text from the input buffer BUFF, one character at a time. If the buffer is empty, it is refilled. The subroutine increments BUFFPTR (line 92) and tests if BUFFPTR=256 (line 93). If it is not, control returns to the point of call. If BUFFPTR=256 then BUFF must be refilled (line 95) and BUFFPTR must be reset to 0 (line 101).

To prevent infinite loops, when symbolic source files are not properly terminated by [END], a test is made on the number of characters read (local variable NCHAR). If it is zero, as would be the case only if a source file lacked proper termination, the program termiantes by calling EXIT (lines 96-100).

5.1.1.2.3.4.2.2 Subroutine NEWWORD

This subroutine sets up the environment to process a new microword. Its functions are:

- isolating the symbolic source text for one microword,
- printing the lines as they are read,
- compressing blank, tab, CR, LF characters and comments out of the symbolic code and placing it into array CODE,
- resetting CODEPTR, PHVECT and FIELD,
- initializing MWORD to the default value,
- resetting of all delayed resolution flags,
- initializing PHRASE(0) to any value other than ';'

From a computer program organization point of view subroutine NEWWORD contains 3 subroutines:

- LINEOUT (lines 108-116)
- LINEADV (lines 117-121)
- CODEADV (lines 122-134)

and its own body of code (lines 135-247).

5.1.1.2.3.4.2.2.1 Subroutines internal to subroutine NEWWORD

5.1.1.2.3.4.2.2.1.1 Subroutine LINEOUT

This subroutine outputs a line to the lineprinter and clears the buffer.

If array LINE contains at least 1 character it is output by means of a call to ALPHA (lines 109-110). The line is ended by calling EOL (line 111).

LINE is set to all blanks (lines 112-114) and LINEPTR is reset to 0 (line 115).

5.1.1.2.3.4.2.2.1.2 Subroutine CODEADV

This subroutine advances the pointer to array CODE, CODEPTR, by one (line 123). Microwords longer than 256 characters cause a terminal error to be flagged. (No legal microword of more than 256 characters is possible anyway). In that case the program is terminated abnormally (lines 119-133).

5.1.1.2.3.4.2.2.2 The body of subroutine NEWWORD

A top level flowchart of this subroutine is shown in Figure 5.1-9.

Resetting CODE and LINE consists of filling both arrays with blanks (lines 135-137) and setting CODEPTR and LINEPTR to zero (lines 138-142).

In the main loop (lines 144-218) the local variable CHAR always contains the latest character from BUFF. Initially it must not be a semicolon, therefore CHAR is set to blank (line 143).

The loop is governed by a DO WHILE CHAR $\neq$ ';' statement (line 144). CHAR is updated from BUFF(BUFFPTR) (line 145) and BUFFPTR is advanced by a call to BUFFADV (line 146).

The series of conditionally executed statements, based on the value of CHAR, begins by a test for CHAR=CR (line 147). If the condition is satisfied, LINE is printed by means of a call to LINEOUT (line 149) and if the next character is an LF then BUFFPTR is advanced passing over the LF (line 150).

The next test is for CHAR=TAB. Tab characters cause LINE to be padded with blanks until the next position that is a multiple of 8 is reached, as if the tab stops where set at 8,16,24, ..., 8n, . . . This is accomplished by the code in lines 155-159.

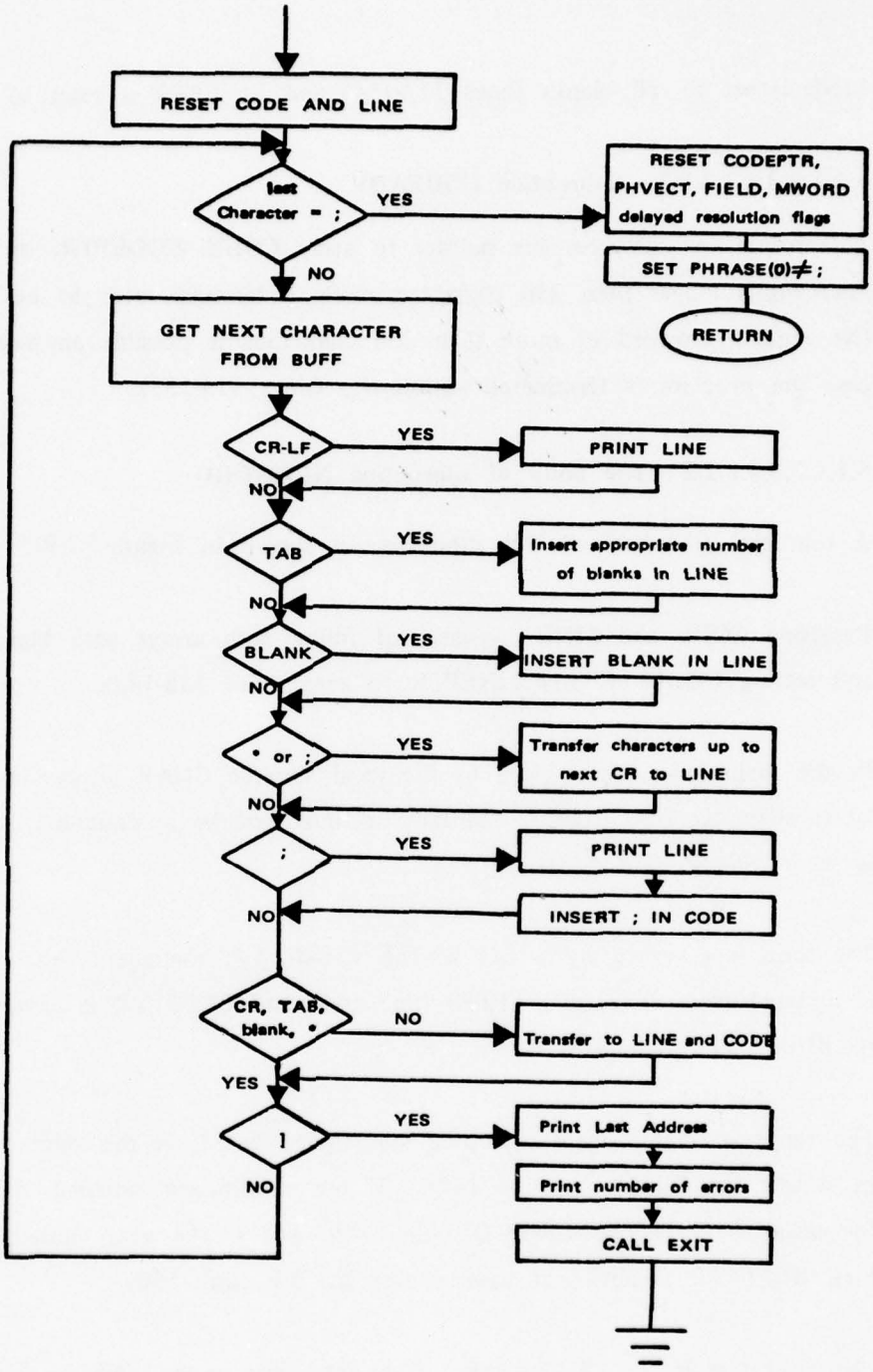


Figure 5.1-9 Flowchart of subroutine NEWWORD.

Next, if CHAR is a blank, LINEPTR is advanced by a call to LINEADV (line 100).

All symbolic code following an asterisk or semicolon is treated as a comment. Following the test for CHAR = '*' OR CHAR = ';' this is done in lines 164-178. The code in lines 170-176 handles tabs embedded in comments. The comment is transfered only to LINE, not to CODE. Thus it is printed in the listing but is ignored by the translation.

If CHAR is a semicolon (line 179), LINE is printed (any comments following it, having been transfered to LINE previously) and the CR-LF pair is stepped over (lines 181-183). Last, the semicolon is entered into CODE (lines 186-187).

All characters othern than CR, TAB, blank, asterisk or semicolon (line 188) are simply entered into both LINE and CODE (lines 190-193).

If CHAR is a] the end of the symbolic microcode has been reached and the program has to proceed to normal termination (line 194). The next available address is printed on the listing (lines 196-203) and the number of errors is printed and displayed on the CRT (lines 204-215). The program terminates by calling EXIT (line 216).

After exit from the main loop, CODEPTR is reset to zero (line 219), PHVECT is set to all zeros (lines 220-222), FIELD is set to all zeros (lines 223-225), MWORD is set to the default value (lines 226-241) the delayed resolution flags are all set to 0 (false) (lines 242-245), and PHRASE (0) is set to a value other than a semicolon (arbitiarily it is set to '9'). Control returns to the point following the point of call from line 247.

5.1.1.2.3.4.2.3 Subroutine NEWPHRASE

Figure 5.1-10 shows a flowchart of subroutine NEWPHRASE. This subroutine transfers the next phrase from CODE to PHRASE. If the character CODEPTR is pointing to in CODE is a semicolon, then PHRASE (0) = ';' is returned (lines 251-254).

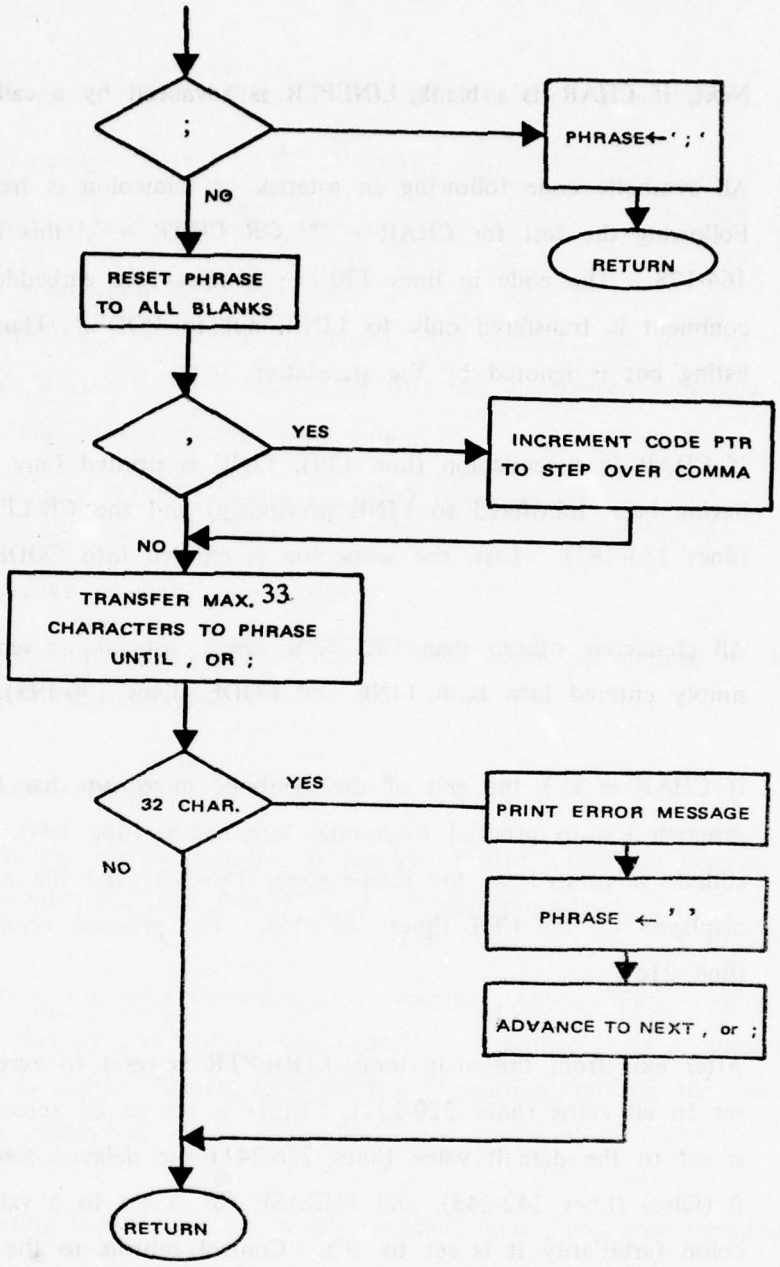


Figure 5.1-10 Flowchart of subroutine NEWPHRASE.

PHRASE is set to all blanks (lines 255). If the character CODEPTR is pointing to in CODE is a comma, CODEPTR is incremented to step over the comma (line 256). Next, up to 33 characters are copied from CODE to PHRASE, until a comma or semicolon is encountered (lines 259-271). Since no legal phrase can be longer than 32 characters, if 33 have been copied, there is an error. In that case PHRASE (0) is set to blank and CODEPTR is advanced to the nearest comma or semicolon (lines 279-282). At the end of the subroutine (line 284) control returns to the point following the point of call.

5.1.1.2.3.4.2.4 Subroutine IDENTIFY

This subroutine sets the variable TYPE according to the contents of the array PHRASE. The first few characters of PHRASE are examined and if the type of phrase contained in PHRASE is recognized, then TYPE is assigned a value in the range 0 to 17 (lines 286-375). If PHRASE(0) is a blank, the implication is that a phrase longer than 32 characters was encountered, and TYPE is set to 18 (lines 376-380). If PHRASE(0) is a semicolon, it means that no further phrases are to be processed in this microword, and TYPE is set to 19 (lines 381-385). If none of the above, then there is an error and the phrase can not be recognized. TYPE is set to 20 (line 386).

If the phrase is a valid recognizable phrase (i.e. TYPE $<$ 18) then PHVECT(TYPE) is checked to see if another phrase of this type has already occured in this microword (lines 387-388). If yes, an error message is printed and TYPE is set to 18, indicating error (lines 389-395). Finally for valid recognizable phrases (i.e. TYPE $<$ 18) PHVECT(TYPE) is set to TRUE (i.e. FF).

5.1.1.2.3.4.2.5 ASCII2BIN Function

This function takes one byte, containing the ASCII representation of a hexadecimal digit, as an argument and returns a 2-byte value. The most significant byte is 0 if the argument represents an ASCII hexadecimal digit, FF otherwise. The least significant byte contains the binary equivalent of the hexadecimal digit.

First the argument, DIGIT, is tested to see if it is a hexadecimal digit (line 403). If the test condition is satisfied, the conversion is performed and the most significant half of the return value is set to 0 (lines 405-410). Otherwise the return value is set to FF00 (lines 411-414).

5.1.1.2.3.4.2.6 Subroutine PRINTROUTINE

This subroutine outputs the microword buffer, MWORD, to the binary output file, prints the control bits in the format shown in Figure 5.1-11 and increments the address counter ADDR.

The greater part of this subroutine is taken up by the formatting of the printed output. A description of these formatting operations is not given here, because the code is very straightforward and constitutes a simpler documentation then a verbal description would.

The output to the binary file is done by means of subroutine ALPHA (line 419). If the RS-bit (bit 4 of the microword) is set then X is printed as the value of both R and T.

5.1.1.2.3.4.3 Initializations at Entry into PASS2

The global variables ADDR and E are initializated to zero (lines 517-518). BUFF and BUFFPTR are initialized by setting BUFFPTR to 255, indicating BUFFER empty, and calling BUFFADV (lines 519-520).

5.1.1.2.3.4.4 The Microcode Translation Loop

This loop encompasses the remainder of the program (lines 521-1893) and is executed once for each microword. Exit from the loop occurs when the program terminates in subroutine NEWWORD, which is called once in each iteration (review Figure 5.1-8).

From a computer program organization point of view, this loop contains:

- A call to subroutine NEWWORD, to read a microword (lines 252);

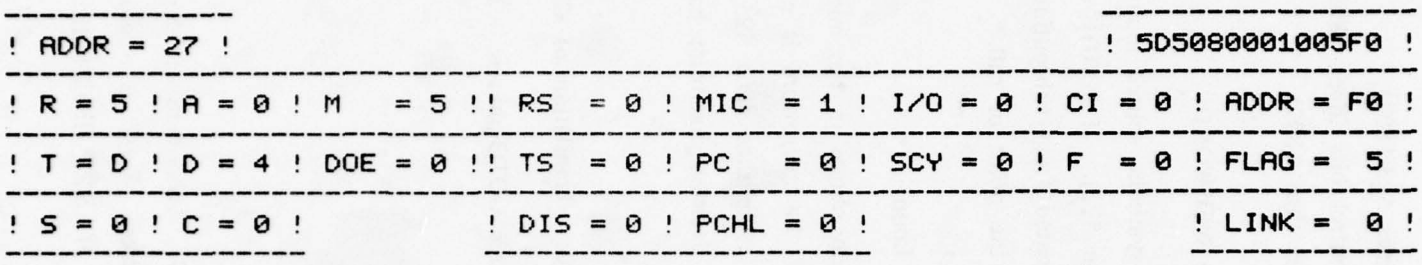


Figure 5.1-11 Microword print format for the Q-80 Microprogram Assembler.

- Phrase translation loop (lines 523-1804);
- Delayed resolution of control bits (lines 1805-1891);
- A call to PRINTROUTINE (line 1892).

5.1.1.2.3.4.4.1 Call to Subroutine NEWWORD

One microword is read from the symbolic source input stream by means of a call to subroutine NEWWORD (line 522). If [END] or the end of file is encountered, the program is terminated from subroutine NEWWORD without returning to the point following the point of call.

5.1.1.2.3.4.4.2 Phrase Translation Loop

This loop is executed once for each phrase in the microword. It is terminated when the semicolon at the end of the microword is encountered. A phrase is read (line 524) and its type is established (line 525). The remainder of the loop is taken up be a DO CASE decision construct based on the value of TYPE (lines 526-1803).

5.1.1.2.3.4.4.2.0 The Case TYPE=0: Translation of GOTO-phrases

The block in lines 527-754 translates GOTO-phrases. There are the following types of GOTO-phrases:

- GOTO < *ddd* + @>
- GOTO *label*
- GOTO *label(suffix)*
- GOTO *label(flags)*

ddd is one of 000,256,512 or 768. *Label* consists of 1-8 alphanueric characters beginning with an alphabetic character. *Suffix* is a group of 1,3,4 or 7 binary digits (0-s or 1-s). *Flags* is one of S,Z,P,HHI,RHHI, or EOPCODE.

Phrases of the type GOTO <*ddd* + @> are translated in lines 532-562. The FS-bits are set to 1111 (lines 536,543,550,557) and the two most significant bits of

the ADDR-field are set to 00 (line 535) for *ddd* = 000, 01 (line 542) for *ddd* = 256, 10 (line 549) for *ddd* = 512, or 11 (line 556) for *ddd* = 768. If one of these 4 phrases has been recognized and translated a branch to END0 is taken (lines 539,546,553,560) bypassing the remainder of the block.

The remaining 3 types of GOTO-phrases all have *label* after the string 'GOTO'. The length of the *label* is established in lines 564-567 and if the label is longer than 8 characters an error message is printed (lines 571-572) and processing of the phrase is terminated by branching to END0 (line 573).

Next it is tested whether the first character is alphabetic (line 575). If it is not, an error message is printed (lines 578-579) and the processing of the phrase is terminated by branching to END0 (line 580).

If the first character is alphabetic, it is copied into the first byte of the local array LABL (line 582).

The remaining characters in *label* are checked for being alphanumeric and copied into array LABL in lines 583-593. If any non-alphanumeric chacter should be encountered, an error message will be printed (lines 588-589) and the processing of the phrase will be terminated by branching to END0 (line 590).

In lines 594-597, the symbol table (array MEMORY) is searched for the entry corresponding to the contents of LABL. If none is found an error message is printed and the processing of the phrase is terminated by branching to END0 (lines 598-604).

If the symbol table entry corresponding to LABL is located, the suffixlength and address are copied from the symbol table to the local variables SUFFIXLENGTH and ADDR.

If SUFFIXLENGTH is zero and there is a *suffix* or *flags* present in the phrase, an error message is printed and the processing of the phrase is terminated by branching to END0 (lines 607-613).

If SUFFIXLENGTH is zero, the 9 MSB of ADDR are copied into the address field of the microword and the FS-bits are set to 0000 if the LSB of address is 0, or to 0001 if the LSB of ADDR is 1 (lines 614-625). Further processing of the phrase is bypassed by branching to END0, translation of the phrase having been successfully accomplished.

Now the only remaining possible GOTO phrases are of the type GOTO *label(suffix)* or GOTO *label(flags)*. The expected location of the parantheses can be computed based on SUFFIXLENGTH. If the parantheses are not where they are expected to be, an error message is printed and the processing of the phrase is terminated by branching to END0 (lines 626-632).

If the parantheses are where expected a DO-CASE-construct is entered (lines 633-753). The cases SUFFIXLENGTH = 1, = 3, = 4, and =7 are processed by different blocks within the DO-CASE-construct (lines 635-659, 661-690, 691-720 and 723-752 respectively).

In the case where SUFFIXLENGTH = 1 there are 5 possible legal characters that can appear between the parantheses: 0, 1, S, Z or P. If the suffix '0' was specified, the FS-bits must be set to 0000 (lines 637-638). If the suffix '1' was specified, the FS-bits must be set to 0001 (lines 639-640). If the flagset 'S' was specified, the FS-bits must be set to 0010 (lines 641-642). If the flagset 'Z' was specified, the FS-bits must be set to 0011 (lines 643-644). If the flagset 'P' was specified, the FS-bits must be set to 0100 (lines 645-646). If none of these was specified, an error message is printed and the processing of the phrase is terminated by branching to END0 (line 647-653). Finally the contents of ADDR are copied to the address field of the microword (lines 654-658).

In the case where SUFFIXLENGTH = 3, there are two alternatives. A 3-digit binary suffix or the characters 'HHI' way appear between the parantheses.

If a 3-digit binary suffix is specified, the value represented by these 3 bits is ORed with ADDR and the address and FS-bits of the microword are set accordingly (lines 663-676). If 'HHI' is specified, the FS-bits are set to 1001 and the 9 MSB of ADDR are copied into the address field of the microword (lines 677-686). If neither of the above was specified, an error message is printed (lines 687-689).

In the case where SUFFIXLENGTH = 4, there are two alternatives. A 4-digit binary suffix or the characters 'RHHI' way appear between the parantheses.

If a 4-digit binary suffix is specified, the value represented by these 4 bits is ORed with ADDR and the address and FS-bits of the microword are set accordingly (lines 693-706). If 'RHHI' is specified, the FS-bits are set to 1000 and the 9 MSB of ADDR are copied into the address field of the microword (lines 707-716). If neither of the above was specified, an error message is printed (lines 717-719).

In the case where SUFFIXLENGTH = 7, there are two alternatives. A 7-digit binary suffix or the characters 'EOPCODE' way appear between the parantheses.

If a 7-digit binary suffix is specified, the value represented by these 7 bits is ORed with ADDR and the address and FS-bits of the microword are set accordingly (lines 725-738). If 'EOPCODE' is specified, the FS-bits are set to 1100 and the 9 MSB of ADDR are copied into the address field of the microword (lines 739-748). If neither of the above was specified, an error message is printed (lines 749-751).

5.1.1.2.3.4.4.3.1 The Case TYPE=1: Translation of Register Assignment Phrases

The block in lines 755-839 translates register assignment phrases. There are two

legal register assignment phrases:

- REG(n) = ALC
- REG(n) = SHIFTER

The register address n can be a hexadecimal digit (e.g., 5 or D), or #. Translation is as follows: M_2 is set to 0 for = SHIFTER and to 1 for = ALC. If the register address is hexadecimal, the R field is set accordingly and RS is set to 0. If the register address is #, RS is set to 1.

The code translating register assignment phrases begins with a test for register address being specified by #. (line 758). If this is the case, a further test has to be made to see if RS has been set to 0 by some other phrase (line 760). If RS has been set to 0 by another phrase, an error message is printed and processing of the phrase is terminated by branching to END1 (lines 761-766).

Otherwise, RS is set to 1 and the local variables REGNO and BADREG are set as if REG(0) had been specified (lines 767-772).

If the register address is not #, REGNO and BADREG are set by a call to the function ASCII2BIN (line 775). A further test has to be made to see if RS has been set to 1 by another phrase, an error message is printed and processing of the phrase is terminated by branching to END1 (lines 777-782) Otherwise RS is set to 0 (lines 783-786).

If an invalid register address has been encountered (line 788), an error message is printed (lines 791-793) and the phrase is ignored by jumping to the end of the block (line 793).

If the R-field is already set (FIELD(13) = FF) to a value other than the register address held in REGNO (line 795), an error message is printed and the phrase is ignored by jumping to the end of the block (lines 796-801).

If = ALC is specified (line 802) and setting M_2 to 1 causes a conflict (line 804), an error message is printed (lines 805-810). If it does not cause a conflict, R is set to the value stored earlier in REGNO (line 813), M_2 is set to 1 (line 815), and FIELD(13) and FIELD(16) are set to FF (line 812 & 814) indicating that the R and M_2 fields have been set. Thereafter, a jump is taken to the end of the block (line 816).

If = SHIFTER is specified, the procedure (lines 819-834) is the same as above (lines 802-818) except M_2 is set to 0.

If the character string following the equal sign is neither = ALC nor = SHIFTER, an error message is printed (lines 835-838). The block ends at line 839.

5.1.1.2.3.4.4.3.2 The Case TYPE=2: Translation of P1B-Phrases

The block in lines 988-1078 translates P1B phrases. There are three legal P1B phrases:

- P1B = REG(n)
- P1B = DI
- P1B = @

The register address n can be a hexadecimal digit (e.g. 5 or D), or #. Translation is as follows: For P1B = DI, S_1 is set to 0 and M cannot take on the value 000. For P1B = REG(n), S_1 is set to 1. The R-field is set to the binary equivalent of the hexadecimal digit specified in the register address, or to zero if #. is specified. The RS-bit is set to 1 if # is specified, and to 0 otherwise.

The code translating P1B phrases begins with a test for the presence of the character string 'REG(' in the array PHRASE (line 843). The block in lines 844-905 is entered if the above test is satisfied. This block sets the R-field to the binary equivalent of the register address specified and sets the RS-bits to 1 if # is specified as the register address. A number of illegal specifications are detected and appropriate messages issued.

If # is specified in the register address (line 845), the RS bit is set to 1 (line 857) and the register address is translated as if REG(0) had been specified (lines 855-856). If the register address is specified by anything other than REGNO and BADREG are set by a call to the function ASCII2BIN (line 862). If an invalid register address has been encountered (line 875) an error message is printed and the phrase is ignored by jumping to the end of the block (lines 876-881).

The code in lines 882-892 sets the R-field. If the R-field is already set by a preceding phrase and is set to a value other than the one contained in REGNO (line 882), an error message is printed and the phrase is ignored by jumping to the end of the P1B-translation block (lines 883-888). Otherwise, the R-field is set to the value of REGNO and FIELD(13) is set to FF, indicating that R has been set (lines 889-892).

The setting of the S-bits is done in lines 893-904. If the phrase is P1B = REG(n) then S_1 is set to 1 (lines 900-904), unless this causes a conflict (tested in line 893). If a conflict would arise, an error message is printed and the phrase is ignored by jumping to the end of the P1B-phrase translation block (lines 894-899).

For the phrase P1B = DI (line 906), the S_1 is set to 0, unless this causes a conflict with previous phrases (tested in line 908). The delayed resolution flag P1BDIPHRASEPRESENT has to be set to insure that M $\neq$ 000. If setting S_1 = 0 causes a conflict an error message is printed and the phrase is ignored by jumping to the end of P1B-phrase translation block (lines 909-914). The phrase P1B = @ is translated like P1B = DI, except the IM-bit is set to 1 (lines 922-938).

If the phrase does not satisfy any one of the three tests (lines 843, 906, 922) then the phrase is illegal and an error message is printed (lines 939-942). The block ends in line 943.

5.1.1.2.3.4.4.3.3 The Case TYPE=3: Translation of P2B-Phrases

The block in lines 1079-1142 translates P2B phrases. There are four legal P2B phrases:

- P2B = P2B
- P2B = DI
- P2B = @
- P2B = REG(n).

The register address n can be an octal digit or #.

Translation is as follows. For P2B = P2B, the A-bits must be 01 and S_1 must be 1. For P2B = REG(n) either $S_1 = 0$ and $A \neq 01$ or $S_1 = 1$ and $A = 01$. The T-field is set to the binary equivalent of the octal digit, or to zero if # was specified. If # was specified, RS must be set to 1; if an octal digit was specified it must be set to 0.

The code translating P2B phrases consists of 4 tests (lines 947, 464, 980, 997) deciding which P2B phrase is at hand. If all four tests are failed, an error message is printed and the phrase is ignored (lines 1043-1046).

If the phrase is P2B = P2B (tested in line 947), a further test is made to see if the control bit settings $S_1 = 0$ and $A = 01$ cause a conflict (line 949). If yes, an error message is printed and the phrase is ignored by jumping to the end of the P2B phrase translation block (lines 950-955). If no conflict arises, the A-bits are set to 01, S_1 is set to 1 and FIELD(7), FIELD(8), and FIELD(11) are set to FF (lines 950-962).

If the phrase is P2B = DI (tested in line 964), a further test is made to see if the control bit setting $S_1 = 1$ and $A \neq 01$ cause a conflict (line 966). If yes, an error message is printed and the phrase is ignored by jumping to the end of the P2B phrase translation block (lines 967-972).

If no conflict arises, the delayed resolution flag NOTAD1 is set preventing A from taking on the value 01, S_1 is set to 1, IM is set to 1 and FIELD(11) is set

to FF (lines 989-995).

If the phrase is P2B = REG(n) (tested in line 997), the register address is decoded first. If it is specified by a # (line 999) then the RS is set to 1 and REGNO and BADREG are set as if REG(0) had been specified (line 1008-1013). If the register address is specified by a hexadecimal digit, its binary equivalent is computed by a call to the function ASCII2BIN (line 1016) and RS is set to 0. If setting RS causes a conflict an error message is printed (lines 1000-1007 and 1017-1023). If the register address is not valid (tested in line 1029) an error message is printed and the phrase is ignored by jumping to the end of the P2B-phrase translation block (lines 1030-1035). If the register address is valid, the value of REGNO is entered into the T-field and FIELD(14) is set to FF (lines 1036-1039).

Since two control bit settings are possible (S_1 = 0, A ≠ 01 and S_1 = 1, A = 01), the delayed resolution flag P2BREGPHRASEPRESENT is set to FF (line 1040).

The P2B-phrase translation block ends after the unrecognized phrase error trap (lines 1043-1046) at line 1047.

5.1.1.2.3.4.4.3.4 The Case TYPE=4: Translation of DO-Phrases

The block in lines 1048-1105 translates the DO-phrases. There are 4 valid DO-phrases:

- DO = OFF
- DO = ALC
- DO = P1B
- DO = P2B.

Translation is as follows: For DO = ALC the M-bits are set such that M ≠ 11x and DOE is set to 1. For DO = P1B the M-bits are 111 and DOE is set to 1. For DO = P2B the M-bits are 110 and DOE is set to 1. For DO = OFF the DOE-bit is set to 0.

The code translating DO-phrases consists of 4 tests (lines 1049, 1054, 1069 and 1085), deciding which DO-phrase is at hand. If all four tests are failed, an error message is printed (lines 1101-1104) and the phrase is ignored.

If the phrase is DO = OFF (tested in line 1049) the DOE is set to 0 (line 1051).

If the phrase is DO = ALC (tested in line 1054) a further test is made to see if M is set to 110 or 111 (lines 1056). If yes, an error message is printed and the phrase is ignored by jumping to the end of the DO-phrase translation block (lines 1057-1062). If no conflict arises, the delayed resolution flag DOALCPHRASEPRESENT is set to FF and DOE is set to 1 (lines 1063-1067).

If the phrase is DO = P1B or DO = P2B, DOE is set to 1 and M is set to 111 or 110, respectively (lines 1078-1083 and 1094-1099). In case of conflicts error messages are printed (lines 1072-1077 and 1088-1093). The DO-phrase translation block ends after unrecognized phrase error trap (lines 1101-1104) at line 1105.

5.1.1.2.3.4.4.3.5 The Case TYPE=5: Translation of ALC-Phrases

The block in lines 1106-1287 translates ALC-phrases. ALC-phrases are syntactically more complex than other phrases. They consist of a left operand which can be:

- P1B
- .NOT.P1B

followed by an operator, which can be:

- +
- OR
- AND

followed by a right operand, which can be:

- 0
- P2B
- .NOT.P2B

optionally followed by a carry-in specification. In addition to these, phrases of the form

ALC = *left* – P2B

are translated as if

ALC = *left* + .NOT.P2B + CO

had been specified.

After some preliminary processing (lines 1107-1136), the carry-in, right operand, left operand and the operator are translated separately (lines 1137-1151, 1152-1204, 1205-1237 and 1238-1286 respectively).

The block begins with a search for the rightmost non-blank character in array PHRASE (lines 1107-1110). The block in lines 1111-1136 handles phrases containing '–P2B' as the operator and right operand. Such phrases, if legally specified, are not longer than 25 characters. Thus, if the last non-blank character of the phrase is to the right of position 24, an error message is printed (lines 1129-1135) and the phrase is ignored by jumping to the end of the ALC-phrase translating block. If the phrase is not too long, the string '+.NOT.P2B+CO' is substituted for '–P2B' (lines 1115-1126) and the pointer to the last non-blank character is changed to reflect the lengthened phrase (line 1127). Note that the test for a phrase containing –P2B that is longer than 25 characters is necessary in order to avoid overflowing the size of array PHRASE when the substitution takes place.

The carry-in part of the phrase is translated in lines 1137-1151. If a carry-in is specified explicitly by '+CO', '+COUT' or '+.NOT.COUT', then the CI field is set to 01, 10 or 11 respectively (lines 1140, 1145, 1150). The pointer to the last non-blank character is adjusted indicating that the carry-in is already processed. If no explicit carry-in specificatin is present, CI is left in its default state, which is 00.

The right operand part of the phrase is translated in lines 1152-1204. The three legal specifications for the right operand (0, .NOT.P2B and P2B) are recognized by

three tests (lines 1152, 1168, and 1184). If all three tests are failed, the right operand is illegally specified. In that case, an error message is printed and the phrase is ignored by jumping to the end of the ALC-phrase translation block (lines 1200-1204).

The translation of 0 as the right operand results in the control bit settings $D_1 = 1$. In line 1154, a test is made to see if this setting causes a conflict with earlier phrases. If there is a conflict, an error message is printed and the phrase is ignored by jumping to the end of the ALC-phrase translation block (lines 1155-1160).

If no conflict occurs, D_1 is set to 1 (line 1163), FIELD(20) is set to FF (line 1162) indicating that D_1 has been set, the pointer is moved one position to the left (line 1164) now pointing to the end of the operator, and a branch is taken to the end of the right operand translating code (line 1165).

The translation of .NOT.P2B as the right operand results in the control bit settings $D_2 = 0$ and $D_1 = 0$. In line 1170 a test is made to see if this setting causes a conflict with earlier phrases. If there is a conflict, an error message is printed and the phrase is ignored by jumping to the end of the ALC-phrase translation block (line 1171-1176).

If no conflict occurs, D_2 and D_1 are set to 0 (line 1179) and FIELD(19) and FIELD(20) is set to FF (line 1178) indicating that D_2 and D_1 have been set. The pointer is moved 8 positions to the left (line 1180) now pointing to the end of the operator. A branch is taken to the end of the right operand translating code (line 1181).

The translation of P2B as the right operand results in the control bit settings $D_2 = 1$ and $D_1 = 0$. In line 1186, a test is made to see if this setting causes a conflict with earlier phrases. If there is a conflict, an error message is printed and the phrase is ignored by jumping to the end of the ALC-phrase translation block (lines 1187-1192).

If no conflict occured D_2 is set to 1, D_1 is set to 0, and FIELD(19) and FIELD(20) are set to FF (lines 1194-1195). The pointer is moved 3 positions to the left (line 1196) now pointing to the end of the operator. A branch is taken to the end of the right operand translating code (line 1197).

The left operand of the phrase is translated in lines 1205-1237. The two legal specifications for the left operand (P1B, and .NOT.P1B) are recognized by two tests (lines 1205 and 1218). If both tests are failed, the left operand is illegally specified. In that case, an error message is printed and the phrase is ignored by jumping to the end of the ALC-phrase translating block (lines 1233-1237).

Two translations of P1B as the left operand are possible: $D_2 = 0$, $D_1 = 0$ or $D_2 = 1$. If $D_2 = 1$ and $D_1 = 0$ or if D_2 is not set then the translation $D_2 = 1$ is chosen. If D_2 and D_1 are both set to 0 then $D_2 = 0$, $D_1 = 0$ is chosen as the translation.

Accordingly, two tests are made to see which translation is applicable (lines 1207 and 1213) and the corresponding bits are set (lines 1208-1212 and 1214-1217). Only one translation of .NOT.P1B as the left operand is possible: $D_2 = 0$, $D_1 = 1$.

If D_2 is set to 1 or D_1 is set to 0 (line 1220) an error message is printed (lines 1221-1226). Otherwise FIELD(19) and FIELD(20) are set to FF, D_2 is set to 0 and D_1 is set to 1 (lines 1227-1231).

Lines 1233-1236 print an error message if the left operand was not recognized as being legal, followed by a branch to the end of the ALC-phrase translation block from line 1237.

The operator of the phrase is translated in lines 1238-1286. The three legal operators (+, OR and AND) are recognized by three tests (lines 1238, 1253, and 1268). If all three tests are failed, the operator is illegally specified. In that case, an error message is printed (lines 1283-1286).

The translation of + as the operator is $A_1 = 0$. In line 1240 a test is made to see if a conflict occurs due to this control bit setting. If there is a conflict, an error message is printed (lines 1241-1246) and the phrase is ignored by jumping to the end of the ALC-phrase translation block.

If no conflict occurs, A_1 is set to 0 and FIELD(7) is set to FF, indicating that A_1 had been set (lines 1247-1251). A branch to the end of the ALC-translating block is taken from line 1250, thus completing the translation of the phrase.

The translation of OR as the operator is A = 11. In line 1255 a test is made, to see if a conflict occurs due to this control bit setting. If there is a conflict, an error message is printed and the phrase is ignored by jumping to the end of the ALC phrase translation block (lines 1256-1261).

If no conflict occurs, A is set to 11 and FIELD(7) and FIELD(8) are set to FF, indicating that A had been set (line 1452). A branch to the end of the ALC-translating block is taken thus completing the translation of the phrase (lines 1262-1266).

The translation of AND as the operator is A = 10. In line 1270 a test is made, to see if a conflict occurs due to this control bit setting. If there is a conflict, an error message is printed and the phrase is ignored by jumping to the end of the ALC-phrase translation block (lines 1271-1276).

If no conflict occurs, A is set to 10 and FIELD(7) and FIELD(8) are set to FF indicating that A had been set. A branch to the end of the ALC-translating block is taken thus completing the translation of the phrase (lines 1277-1281).

5.1.1.2.3.4.4.3.6 The Case TYPE=6: Translation of SHIFTER-Phrases

The block in lines 1288-1467 translates SHIFTER-phrases. The valid SHIFTER-phrases and their translation are shown in 5.1-7.

Table 5.1-7 Translation of SHIFTER-Phrases

SHIFTER = @	$M_1 = 0, M_0 = 0, S_1 = 0, IM = 1$
SHIFTER = DI,	$M_1 = 0, M_0 = 0, S_1 = 0$
SHIFTER = ALC(6-0)\\ 0,	$M_1 = 0, M_0 = 1, C = 011.$*
SHIFTER = ALC(6-0)\\ 1,	$M_1 = 0, M_0 = 1, C = 010.$*
SHIFTER = ALC(6-0)\\ MXL0IN,	$M_1 = 0, M_0 = 1.$
SHIFTER = 0\\ALC(7-1),	$M_1 = 1, M_0 = 0, C = 011.$*
SHIFTER = 1\\ALC(7-1),	$M_1 = 1, M_0 = 0, C = 010.$*
SHIFTER = MXH0IN\\ALC(7-1),	$M_1 = 1, M_0 = 0.$
SHIFTER = 00\\ALC(7-2),	$M_1 = 1, M_0 = 1, C = 011.$*
SHIFTER = 11\\ALC(7-2),	$M_1 = 1, M_0 = 1, C = 010.$*
SHIFTER = MXH1IN\\MXH0IN\\ALC(7-2).	$M_1 = 1, M_0 = 1.$

The code translating SHIFTER-phrases consists of 11 tests (lines 1284, 1306, 1322, 1338, 1354, 1369, 1385, 1401, 1416, 1432, and 1448) deciding which SHIFTER phrase is at hand. If all 11 tests are failed, an error message is printed (lines 1463-1466).

The blocks entered when any one of the 11 tests is passed are all very similar. Each begins with a tests for conflicts with the control bit configuration which constitutes the translation of the respective phrase. If there is a conflict, an error message is printed and the phrase is ignored by jumping to the end of the SHIFTER-phrase translation block.

If there is no conflict, the respective bits are set (as shown in table 3) and the respective elements of array FIELD are set to FF.

---

* Only C_1 and C_0 are actually represented in the microwords. C_2 is always 0.

5.1.1.2.3.4.4.3.7 The Case TYPE=7: Translation of MXL0OUT-Phrases

The block in lines 1468-1491 translates MXL0OUT phrases. There is only one valid MXL0OUT phrase.

- MXL0OUT = ALC(0).

Translation is $M_1 = 0$, $M_0 = 1$. A test is made to see if this control bit setting causes a conflict (line 1471). If there is a conflict, an error message is printed and the phrase is ignored by jumping to the end of the MXL0OUT-phrase translating block (lines 1472-1477).

If no conflict occurs, the control bits M_1 and M_0 are set to 0 and 1, respectively and FIELD(16) and FIELD(17) are set to FF indicating that M_1 and M_0 have been set. (lines 1478-1483).

If an invalid MXL0OUT-phrase is specified (anything other than MXL0OUT = ALC(0), tested in line 1469), an error message is printed (lines 1485-1490) and the phrase is ignored.

5.1.1.2.3.4.4.3.8 The Case TYPE=8: Translation of MXL1OUT-Phrases

The block in lines 1492-1514 translates MXL1OUT-phrases. There is one legal MXL1OUT-phrase:

- MXL1OUT = ALC(1)

Translation is M = 011. A test is made to see if the control bit setting M = 011 causes a conflict (line 1495). If there is a conflict, an error message is printed and the phrase is ignored by jumping to the end of the MXL1OUT-phrase translating block (lines 1496-1501).

If no conflict occurs, M is set to 011 and FIELD(16), FIELD(17), and FIELD(18) are set to FF, indicating that the M-bits have been set (lines 1502-1506).

If an invalid MXL1OUT-phrase is specified (anything other than MXL1OUT = ALC(1)) an error message is printed (lines 1508-1513) and the phrase is ignored.

5.1.1.2.3.4.4.3.9 The Case TYPE=9: Translation of MXH0OUT-Phrases

The block in lines 1515-1537 translates MXH0OUT-phrases. There is only one valid MXH0OUT phrase:

- MXH0OUT = ALC(7).

The translation is M = 001. A test is made to see if the control bit setting M = 001 causes a conflict (line 1518). If there is a conflict, an error message is printed and the phrase is ignored by jumping to the end of the MXH0OUT-phrase translating block (lines 1519-1524).

If no conflict occurs, M is set to 001 and FIELD(16), FIELD(17), and FIELD(18) are set to FF, indicating that the M-bits have been set (lines 1525-1529).
If an invalid MXH0OUT-phrase is specified (anything other than MXH0OUT = ALC(7)) an error message is printed (lines 1531-1536) and the phrase is ignored.

5.1.1.2.3.4.4.3.10 The Case TYPE = 10: Translation of MXH1OUT-Phrases

The block in lines 1841-1891 translates MXH1OUT-phrases. Ther is one legal MXH1OUT-phrase:

- MXH1OUT = OFL

Translation is C_1 = 1. A test is made to see if the control bit setting C_1 = 1 causes a conflict (line 1541). If there is a conflict, an error message is printed and the phrase is ignored by jumping to the end of the MXH1OUT-phrase translating block (lines 1542-1546).

If no conflict occurs, C_1 is set to 1 and FIELD(2) set to FF, indicating that C_1 has been set (lines 1547-1550).

If an invalid MXH1OUT-phrase is specified (anything other than MXH1OUT = OFL) an error message is printed (lines 1552-1557) and the phrase is ignored.

5.1.1.2.3.4.4.3.11 The Case TYPE=11: Translation of CO-Phrases

The block in lines 1559-1574 translates CO-phrases. There are two legal CO-phrases:

- CO = GPUCOUT
- CO = L2

The former is translated by setting the CO-bit to 0, the latter by setting it to 1.

Two tests are made to see which of these phrases is at hand (lines 1560 and 1565). If GPUCOUT is specified, the CO-bit is set to 0 and a branch to the end of the block is taken (lines 1561-1564). If L2 is specified, the CO-bit is set to 1, and a branch to the end of the block is taken (lines 1566-1569). Illegal CO-phrases are trapped by printing an error message (line 1570-1573).

5.1.1.2.3.4.4.3.12 The Case TYPE=12: Translation of MFLAGS-Phrases

The block in lines 1575-1600 translates MFLAGS-phrases. There are four legal MFLAGS-phrases:

- MFLAGS = DISABLE
- MFLAGS = ENABLE(OUT)
- MFLAGS = ENABLE(SZP)
- MFLAGS = ENABLE(ALL)

The first of these is translated by setting the MFLAGS-bits to 00, the second by setting them to 01, the third by setting them to 10, and the fourth by setting them to 11.

Four tests are made to see which of these phrases is at hand (lines 1576, 1581, 1586, and 1591). If a phrase is recognized by one of these tests, the MFLAGS-bits are set accordingly and a branch to the end of the block is taken (lines 1577-1580, 1582-1585, 1587-1590, 1592-1595). If the phrase is not recognized as one of these four, an error message is printed (lines 1596-1599).

5.1.1.2.3.4.4.3.13 The Case TYPE=13: Translation of COUT-Phrases

The block in lines 1601-1626 translates COUT-phrases. There are four legal COUT-phrases

- COUT = COUT
- COUT = GPUCOUT
- COUT = DO(7)
- COUT = DO(0)

The first of these is translated by setting the CO-bits to 00, the second by setting them to 01, the third by setting them to 10, and the fourth by setting them to 11.

Four tests are made to see which of these phrases is at hand (lines 1602, 1607, 1612 and 1617). If a phrase is recognized by one of these tests, the CO-bits are set accordingly and a branch to the end of the block is taken (lines 1603-1606, 1608-1611, 1613-1616, 1618-1621). If the phrase is not recognized as one of these four, an error message is printed (lines 1622-1625).

5.1.1.2.4.3.3.4.14 The Case TYPE=14: Translation of the NOLOAD Phrase

The block in lines 1627-1638 translates the NOLOAD phrase:

- NOLOAD

Translation consists of setting M = 100. If this causes a conflict (tested in line 1628) an error message is printed (lines 1629-1633). Otherwise M is set to 100 and FIELD(16), FIELD(17), and FIELD(18) are set to FF, indicating that the M-bits have been set (lines 1634-1637).

5.1.1.2.3.4.4.3.15 The Case TYPE=15: Translation of I/O-Phrases

The block in lines 1639-1719 translates I/O phrases. There are 15 possible I/O phrases. The phrases and their translation are as follows:

PHRASE	TRANSLATION
I/O = .NOT.MEMR	I/O = 0001
I/O = .NOT.MEMW	I/O = 0010
I/O = .NOT.IOR	I/O = 0011
I/O = .NOT.IOW	I/O = 0100
I/O = HLDA	I/O = 0101
I/O = WAIT	I/O = 0110
I/O = INTES	I/O = 0111
I/O = .NOT.INTA.NOT.IFCH	I/O = 1000
I/O = HLDA.WAIT	I/O = 1001
I/O = INTER	I/O = 1010
I/O = .NOT.MEMR.NOT.IFCH	I/O = 1011
I/O = .NOT.MEMR.NOT.STACK	I/O = 1100
I/O = .NOT.MEMW.NOT.STACK	I/O = 1101
I/O = EI	I/O = 1110
I/O = DI	I/O = 1111

The code translating I/O-phrases consists of 15 tests deciding which I/O-phrase is at hand. If all 15 tests are failed, an error message is printed (lines 1715-1718). The blocks entered when anyone of the 15 tests is passed are nearly identical. In each the I/O-bits are set to the corresponding value and a branch to the end of the block (i.e. to END15) is taken.

5.1.1.2.3.4.4.3.16 The Case TYPE=16: Translation of GUA-phrases

The block in lines 1720-1745 translates GUA-phrases. There are four legal GUA-phrases:

- GUA = ENABLE(CE)

- GUA = ENABLE(LE1)
- GUA = ENABLE(LE2)
- GUA = ENABLE(LE3)

The first of these is translated by setting the GUA-bits to 00, the second by setting them to 01, the third by setting them to 10, the fourth by setting them to 11.

Four tests are made to see which of these phrases is at hand (lines 1721, 1726, 1731, and 1736). If a phrase is recognized by one of these tests, the GUA-bits are set accordingly and a branch to the end of the block (i.e. to END16) is taken (lines 1722-1725, 1727-1730, 1732-1735 and 1737-1740). If the phrase is not recognized as one of these four, an error message is printed (lines 1741-1744).

5.1.1.2.3.4.4.3.17 The Case TYPE=17: Translation of E-Phrases

The block in lines 1746-1761 translates E-phrases. There are two legal E-phrases:

- E = 0
- E = 1

Translation consists of setting the E-bit in the microword to 0 or 1, as specified. Two tests are made to see which of the phrases is at hand (lines 1747-1452). In either case the E-bit is set and a branch to END17 is taken (lines 1748-1751 and 1753-1756). If the phrase has not been recognized an error message is printed (lines 1757-1760).

5.1.1.2.3.4.4.3.18 The Case TYPE=18: Errors

The null-statement in line 1762 is executed if a phrase is too long or if a phrase of its type has already been executed (see sections 3.4.2.3 and 3.4.2.4).

5.1.1.2.3.4.4.3.19 The Case TYPE=19: End of Microword

The null-statement in line 1763 is executed if PHRASE (0) = ';', i.e., if the semicolon ending the microword is encountered (see sections 3.4.2.4. and 3.4.4.3).

5.1.1.2.3.4.4.3.20 The Case TYPE=20: Other Phrases

The block in lines 1764-1802 handles the phrases causing explicit setting of control bits and traps unrecognizable phrases (where the text to the left of the equal sign is not valid). There are 5 valid phrases in this category:

- A = x
- C = x
- D = x
- M = x
- S = x where x is a digit from 0 to 7.

The translation is obvious; the respective field is set to the value given.

If the phrase is not one of the 5 shown above (tested in line 1765) an error message is printed (lines 1766-1771).

If the phrase is valid, the value is converted from ASCII to binary (line 1774) and 5 tests are made, to see which control bits are to be set (lines 1775, 1780, 1785, 1791, 1796). In each case, the respective control bits are set and the corresponding elements of array FIELD are set. Note that no tests are made to check if the respective control bits had been set by earlier phrases which causes these explicit control bit settings to override symbolic specifications.

5.1.1.2.3.4.4.4 Delayed Resolution of Control Bits

Delayed resolution of control bits takes place in 2 steps:

- delayed resolution of S_1 and of the A-bits (lines 1805-1847)
- delayed resolution of the M-bits (lines 1848-1895)

5.1.1.2.3.4.4.4.1 Delayed Resolution of S_1 and of the A-Bits

Two delayed resolution flags affect S_1 and the A-bits: P2BREGPHRASEPRESENT and NOTAD1.

If P2BREGPHRASEPRESENT is FF, two translations are possible:

- $S_1 = 1$ and $A = 01$, or
- $S_1 = 0$ and $A \neq 01$.

If NOTAD1 is FF, A may not be set to 01.

Eight conditions determine the value of S_1 and of the A-bits:

1. P2BREGPHRASEPRESENT,
2. NOTAD1,
3. FIELD(11) (indicates S_1 is set),
4. the current value of S_1,
5. FIELD(7) (indicates A_1 is set),
6. FIELD(8) (indicates A_0 is set),
7. the current value of A_1,
8. the current value of A_0.

Figure 5.1-12 shows the value of S_1 and A as a function of these 8 conditions. The array TBL is the computer representation of the Table in Figure 5.1-12 and S_1, A and unresolvable conflicts are determined by table-lookup.

In lines 1807-1822, the variable INDEX is set up according to the eight conditions ennumerated above. The lookup takes place in line 1824. If the table entry looked up corresponds to one of the blank squares in Figure 5.1-12 (tested in line 1825), an error message is printed (lines 1826-1832). This should never happen, unless there is an error in the Q-80 Microprogram Assembler.

If the table entry looked up corresponds to one of the E-squares in Figure 5.1-12 (tested in line 1833), an unresolvable conflict exists and an error message is printed (lines 1834-1838). Otherwise, S_1 and A are set according to the table entry looked up and FIELD(7), FIELD(8), and FIELD(11) are set to FF, indicating that S_1 and A have been set (lines 1839-1846).

5.1.1.2.3.4.4.4.2 Delayed Resolution of M-Bits

Two delayed resolution flags affect the M-bits: DOALCPHRASEPRESENT and P1BDIPHRASEPRESENT. The former is resolved in lines 1848-1868, the latter in lines 1869-1891.

For DOALCPHRASEPRESENT it has to be insured that M $\neq$ 11x. In doing so, one must take care that M not be set to 100, unless a NOLOAD phrase was specified.

Processing of the DOALCPHRASEPRESENT condition begins by checking if M_2 is set to 1 (line 1850). If it is not, M_2 is set to 0 and FIELD (16) is set to FF (lines 1868-1871). If M_2 is set to 1, a further check is made to see

P2BREGPHRASEPRESENT, NOTAD1, FIELD(11), S_1

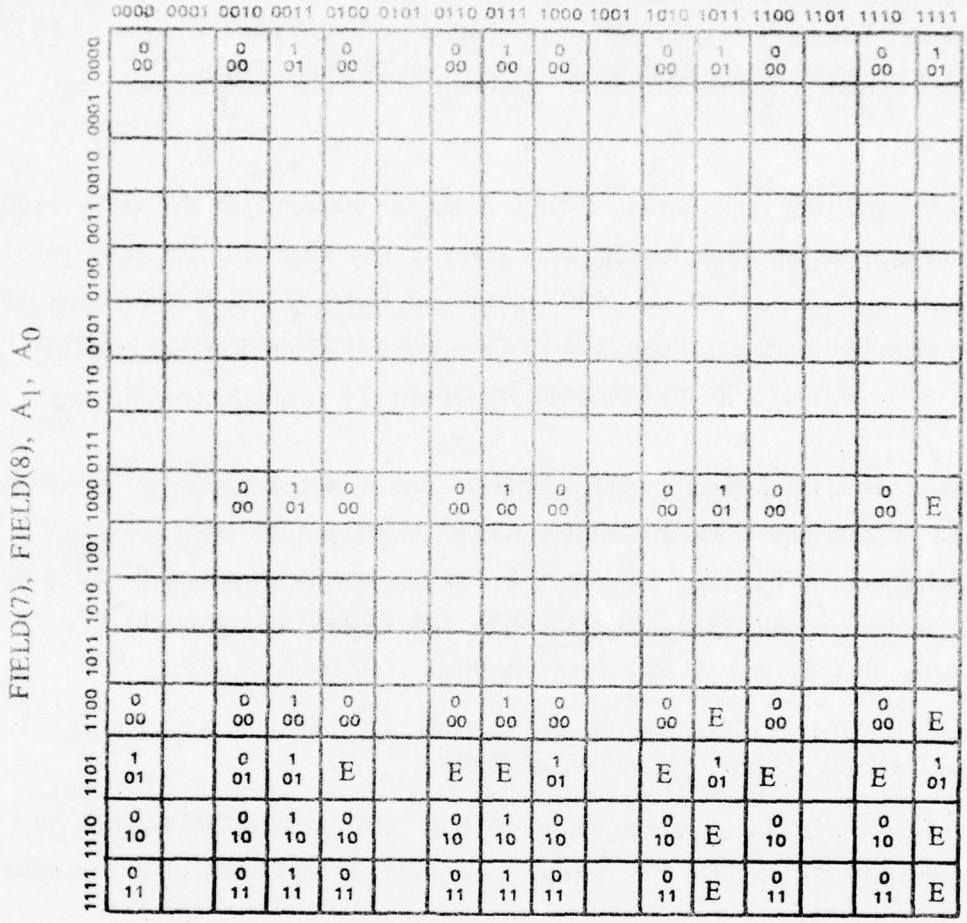


Figure 5.1-12 Delayed resolution of S_1 and of the A-bits. The upper digit is the value of S_1, the 2 lower digits are the value of A_1 and A_0. Blank squares represent conditions that are not expected to occur. E indicates a conflict that can not be resolved.

if M_1 is also set to 1 (line 1852). If it is, an error message is printed (lines 1858-1860). This however may lead to M being inadvertently set to 100. Therefore, if M_0 is not set, it is now set to 1 (lines 1861-1865).

For P1BDIPHRASEPRESENT, it has to insure that M will not be 000. If M has a value other than 000 (line 1875) then there is nothing to be done. If however, M = 000 (line 1875) and FIELD(16), FIELD(17), and FIELD(18) are FF (line 1877) indicating that M has been set to 000 by previous phrases, then an error message is printed (lines 1878-1882).

If M = 000 and not all of FIELD(16), FIELD(17), and FIELD(18) are FF, then M_1 or M_0 or both are not set. (Since the default for M_2 is 1, it could not be 0 without having been set). Therefore, if FIELD (17) is not FF (lines 1884) then M_1 is set to 0 (lines 1885-1888), otherwise M_0 is set to 0 (lines 1889-1892).

5.1.1.2.3.4.4.5 Printing and Output of the Translated Microword

The microword is printed and written to the binary output file by a call to PRINTROUTINE (line 1896).

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE MICROX
OBJECT MODULE PLACED IN :F1:MICROX.OBJ
COMPILER INVOKED BY:  PLM80 :F1:MICROX.PLM DATE(18 OCT 77)


                $PRINT(:LP:) PAGEWIDTH(80) TITLE('Q-80 MICROPROGRAM ASSEMBLER')
  1             MICROX: DO;
                $INCLUDE(INCLUD.DCL)
  2    1   =    OPEN:    PROCEDURE(AFTPTR,FNPTR,ACCCODE,ECHOAFT,STATUS) EXTERNAL;
  3    2   =               DECLARE(AFTPTR,FNPTR,ACCCODE,ECHOAFT,STATUS) ADDRESS;
  4    2   =             END OPEN;
  5    1   =    CLOSE:   PROCEDURE(AFT,STATUS) EXTERNAL;
  6    2   =               DECLARE(AFT,STATUS) ADDRESS;
  7    2   =             END CLOSE;
  8    1   =    READ:    PROCEDURE(AFT,BUFPTR,LENGTH,COUNTPTR,STATUS) EXTERNAL;
  9    2   =               DECLARE(AFT,BUFPTR,LENGTH,COUNTPTR,STATUS) ADDRESS;
 10    2   =             END READ;
 11    1   =    WRITE:   PROCEDURE(AFT,BUFPTR,LENGTH,STATUS) EXTERNAL;
 12    2   =               DECLARE(AFT,BUFPTR,LENGTH,STATUS) ADDRESS;
 13    2   =             END WRITE;
 14    1   =    SEEK:    PROCEDURE(AFT,OPER,BLOCKNOPTR,BYTENOPTR,STATUS) EXTERNAL;
 15    2   =               DECLARE(AFT,OPER,BLOCKNOPTR,BYTENOPTR,STATUS) ADDRESS;
 16    2   =             END SEEK;
 17    1   =    DELETE:  PROCEDURE(AFT,STATUS) EXTERNAL;
 18    2   =                 DECLARE(AFT,STATUS) ADDRESS;
 19    2   =               END DELETE;
 20    1   =    ERROR:   PROCEDURE(ERRNO,STATUS) EXTERNAL;
 21    2   =               DECLARE(ERRNO,STATUS) ADDRESS;
 22    2   =             END ERROR;
 23    1   =    EXIT:    PROCEDURE EXTERNAL;
 24    2   =             END EXIT;
           =
 25    1   =    ALPHA:   PROCEDURE(AFT,LENGTH,BUFPTR) EXTERNAL;
 26    2   =               DECLARE(AFT,LENGTH,BUFPTR) ADDRESS;
 27    2   =             END ALPHA;
 28    1   =    BINARY:  PROCEDURE(AFT,LENGTH,BITE) EXTERNAL;
 29    2   =               DECLARE AFT ADDRESS, (LENGTH,BITE) BYTE;
 30    2   =             END BINARY;
 31    1   =    HEX:     PROCEDURE(AFT,LENGTH,BUFPTR) EXTERNAL;
 32    2   =               DECLARE(AFT,LENGTH,BUFPTR) ADDRESS;
 33    2   =             END HEX;
 34    1   =    INTEG:   PROCEDURE(AFT,BITE) EXTERNAL;
 35    2   =               DECLARE AFT ADDRESS, BITE BYTE;
 36    2   =             END INTEG;
 37    1   =    INTEGD:  PROCEDURE(AFT,TWOBITES) EXTERNAL;
 38    2   =               DECLARE(AFT,TWOBITES) ADDRESS;
 39    2   =             END INTEGD;
 40    1   =    PAGE:    PROCEDURE(AFT) EXTERNAL;
 41    2   =               DECLARE AFT ADDRESS;
 42    2   =             END PAGE;
 43    1   =    PPAGE:   PROCEDURE(AFT) EXTERNAL;
 44    2   =               DECLARE AFT ADDRESS;
 45    2   =             END PPAGE;
 46    1   =    SKIP:    PROCEDURE(AFT) EXTERNAL;
 47    2   =               DECLARE(AFT) ADDRESS;
 48    2   =             END SKIP;
```

```
 49   1  =   SPACE:  PROCEDURE(AFT,LENGTH) EXTERNAL;
 50   2  =             DECLARE(AFT,LENGTH) ADDRESS;
 51   2  =           END SPACE;
           =
 52   1  =   EQUAL:  PROCEDURE(P1,P2,L) BYTE EXTERNAL;
 53   2  =             DECLARE(P1,P2) ADDRESS, L BYTE;
 54   2  =           END EQUAL;
 55   1  =   SETBIT: PROCEDURE(P,I,L,V) EXTERNAL;
 56   2  =             DECLARE P ADDRESS, (I,L,V) BYTE;
 57   2  =           END SETBIT;
 58   1  =   EQUBIT: PROCEDURE(P,I,L,V) BYTE EXTERNAL;
 59   2  =             DECLARE P ADDRESS, (I,L,V) BYTE;
 60   2  =           END EQUBIT;
```

```
                $EJECT
                /*****************************************************************************/
                /*                                                                           */
                /*      DECLARATION OF PUBLIC VARIABLES AND PROCEDURES                       */
                /*                                                                           */
                /*****************************************************************************/

 61   1             DECLARE (SI,BO,LP) ADDRESS PUBLIC,
                             KB  ADDRESS PUBLIC DATA(1),
                             CRT ADDRESS PUBLIC DATA(0);
 62   1             DECLARE CR  BYTE PUBLIC DATA(0DH);
 63   1             DECLARE LF  BYTE PUBLIC DATA(0AH);
 64   1             DECLARE TAB BYTE PUBLIC DATA(09H);
 65   1             DECLARE E BYTE PUBLIC;
 66   1             DECLARE ENDSYMBTAB ADDRESS PUBLIC;
 67   1             DECLARE FIRSTAVAILADDR ADDRESS PUBLIC;
 68   1             DECLARE PAGENO ADDRESS PUBLIC;
 69   1             DECLARE LINENO BYTE PUBLIC;
 70   1             HEADER:PROCEDURE(SW) PUBLIC;
 71   2                     DECLARE SW BYTE;
 72   2                     IF SW THEN CALL PPAGE(LP);
 74   2                           ELSE CALL PAGE(LP);
 75   2                     CALL ALPHA(LP,-20,.(0EH,'          QUESTRON'));
 76   2                     CALL SKIP(LP);
 77   2                     CALL SKIP(LP);
 78   2                     CALL ALPHA(LP,30,.(0EH,
                                   ' Q-80 MICROPROGRAM ASSEMBLER '));
 79   2                     CALL SKIP(LP);
 80   2                     CALL SPACE(LP,70);
 81   2                     CALL ALPHA(LP,4,.('PAGE'));
 82   2                     CALL INTEGD(LP,PAGENO);
 83   2                     CALL SKIP(LP);
 84   2                     CALL ALPHA(LP,41,.(0EH,'-----------------------',
                                                 '-----------------------'));
 85   2                     CALL SKIP(LP);
 86   2                     CALL SKIP(LP);
 87   2                     LINENO = 5;
 88   2                     PAGENO = PAGENO + 1;
 89   2                     END HEADER;
 90   1             EOL:    PROCEDURE PUBLIC;
 91   2                     LINENO = LINENO + 1;
 92   2                     CALL SKIP(LP);
 93   2                     IF LINENO <= 61 THEN RETURN;
 95   2                                     ELSE CALL HEADER(0);
 96   2                     END EOL;
 97   1             PASS1:  PROCEDURE EXTERNAL;
 98   2                     END PASS1;
 99   1             PASS2:  PROCEDURE EXTERNAL;
100   2                     END PASS2;
```

AD-A063 004 QUESTRON CORP SAN DIEGO CALIF F/G 9/2
RADIATION HARDENED MICROPROCESSOR. VOLUME II.(U)
MAY 78 V V NICKEL, P A ROSENBERG F33615-77-C-1001
UNCLASSIFIED AFAL-TR-78-55-VOL-2 NL

2 OF 4
AD
AD63 00 A

```
                $EJECT
                /*****************************************
                /*
                /*      PROLOGUE ACTIVITIES:
                /*
                /*                - DIALOG WITH CONS
                /*                - OPEN SOURCE FILE
                /*                - OPEN BINARY FILE
                /*                - OPEN LINEPRINTER
                /*
                /*****************************************

101   1           DO;
102   2           DECLARE LINE(256) BYTE;
103   2           DECLARE FN(15) BYTE;    /* FN
104   2           DECLARE LENGTH ADDRESS;
105   2           DECLARE (LINEPTR,FNPTR,I) BY
106   2           DECLARE (STATUS,ST,DATEAFT)
107   2           DECLARE DATE(11) BYTE;

                /* READ REMAINDER OF COMMAND L
108   2           CALL READ(KB,.LINE,255,.LENG
109   2           IF LENGTH=2
                        THEN DO;      /* PROM
111   3                     CALL ALPHA(CRT,
112   3                     CALL READ(KB,.L
113   3                     IF LENGTH=2 THE
115   3                     END;

                /* IGNORE LEADING BLANKS */
116   2           LINEPTR = 0;
117   2           DO WHILE LINE(LINEPTR)=' ';
118   3           LINEPTR = LINEPTR + 1;
119   3           END;

                /* COPY FILENAME FROM LINE TO
120   2           FNPTR = 0;
121   2           DO WHILE LINE(LINEPTR)<>' '

122   3           FN(FNPTR) = LINE(LINEPTR);
123   3           LINEPTR = LINEPTR + 1;
124   3           FNPTR   = FNPTR   + 1;
125   3           END;

                /* FILL IN REMAINDER OF FN WIT
126   2           DO I = FNPTR TO 14;
127   3           FN(I) = ' ';
128   3           END;

                /* OPEN SOURCE FILE AND CHECK
129   2           CALL OPEN(.SI,.FN,1,0,.STATU
130   2           IF STATUS=4 OR STATUS=5 OR S
                        THEN DO;
132   3                     CALL ALPHA(CRT,
133   3                     CALL ERROR(STAT
134   3                     CALL EXIT;
135   3                     END;
```

```
 136   2              IF STATUS=13
                             THEN DO;
 138   3                        CALL ALPHA(CRT,20,.('FILE DOES NOT EXIST '));
 139   3                        CALL ERROR(STATUS,.ST);
 140   3                        CALL EXIT;
 141   3                        END;

                    /* OPEN :LP: FILE, PRINT HEADER, FILENAME AND DATE. */

 142   2              CALL OPEN(.LP,.(':LP: '),2,0,.STATUS);
 143   2              PAGENO = 1;
 144   2              CALL HEADER(1);
 145   2              CALL SPACE(LP,8);
 146   2              CALL ALPHA(LP,14,.FN);
 147   2              CALL SPACE(LP,20);
 148   2              CALL OPEN(.DATEAFT,.('DATE '),1,0,.STATUS);
 149   2              CALL READ(DATEAFT,.DATE,11,.LENGTH,.STATUS);
 150   2              CALL CLOSE(DATEAFT,.STATUS);
 151   2              CALL ALPHA(LP,11,.DATE);
 152   2              CALL EOL;
 153   2              CALL EOL;
 154   2              CALL ALPHA(LP,41,.(0EH,'----------------------',
                                           '----------------------'));
 155   2              CALL EOL;
 156   2              CALL EOL;

                    /* CHANGE FILE EXTENSION TO .BIN */
 157   2              FNPTR = 1;
 158   2              DO WHILE FN(FNPTR)<>'.' AND FN(FNPTR)<>' ';
 159   3              FNPTR = FNPTR + 1;
 160   3              END;
 161   2              FN(FNPTR  ) = '.';
 162   2              FN(FNPTR+1) = 'B';
 163   2              FN(FNPTR+2) = 'I';
 164   2              FN(FNPTR+3) = 'N';
 165   2              FN(FNPTR+4) = ' ';

                    /* OPEN BINARY MICROPROGRAM FILE */
 166   2              CALL OPEN(.BO,.FN,3,0,.STATUS);
 167   2              END;

 168   1              CALL PASS1;      /* BUILD SYMBOL TABLE */
 169   1              CALL PASS2;      /* BUILD BINARY FILE  */
 170   1              END MICROX;


MODULE INFORMATION:

     CODE AREA SIZE     = 03FEH   1022D
     VARIABLE AREA SIZE = 0134H    308D
     MAXIMUM STACK SIZE = 0008H      8D
     214 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE MICRO1
OBJECT MODULE PLACED IN :F1:MICRO1.OBJ
COMPILER INVOKED BY:  PLM80 :F1:MICRO1.PLM DATE(25 OCT 77)


                $PRINT(:LP:) PAGEWIDTH(80)
                $TITLE('PASS I OF 0-80 MICROPROGRAM ASSEMBLER')
  1             MICRO1: DO;
                $INCLUDE(INCLUD.DCL)
  2   1   =     OPEN:   PROCEDURE(AFTPTR,FNPTR,ACCCODE,ECHOAFT,STATUS) EXTERNAL;
  3   2   =               DECLARE(AFTPTR,FNPTR,ACCCODE,ECHOAFT,STATUS) ADDRESS;
  4   2   =             END OPEN;
  5   1   =     CLOSE:  PROCEDURE(AFT,STATUS) EXTERNAL;
  6   2   =               DECLARE(AFT,STATUS) ADDRESS;
  7   2   =             END CLOSE;
  8   1   =     READ:   PROCEDURE(AFT,BUFPTR,LENGTH,COUNTPTR,STATUS) EXTERNAL;
  9   2   =               DECLARE(AFT,BUFPTR,LENGTH,COUNTPTR,STATUS) ADDRESS;
 10   2   =             END READ;
 11   1   =     WRITE:  PROCEDURE(AFT,BUFPTR,LENGTH,STATUS) EXTERNAL;
 12   2   =               DECLARE(AFT,BUFPTR,LENGTH,STATUS) ADDRESS;
 13   2   =             END WRITE;
 14   1   =     SEEK:   PROCEDURE(AFT,OPER,BLOCKNOPTR,BYTENOPTR,STATUS) EXTERNAL;
 15   2   =               DECLARE(AFT,OPER,BLOCKNOPTR,BYTENOPTR,STATUS) ADDRESS;
 16   2   =             END SEEK;
 17   1   =     DELETE: PROCEDURE(AFT,STATUS) EXTERNAL;
 18   2   =                 DECLARE(AFT,STATUS) ADDRESS;
 19   2   =               END DELETE;
 20   1   =     ERROR:  PROCEDURE(ERRNO,STATUS) EXTERNAL;
 21   2   =               DECLARE(ERRNO,STATUS) ADDRESS;
 22   2   =             END ERROR;
 23   1   =     EXIT:   PROCEDURE EXTERNAL;
 24   2   =             END EXIT;
          =
 25   1   =     ALPHA:  PROCEDURE(AFT,LENGTH,BUFPTR) EXTERNAL;
 26   2   =               DECLARE(AFT,LENGTH,BUFPTR) ADDRESS;
 27   2   =             END ALPHA;
 28   1   =     BINARY: PROCEDURE(AFT,LENGTH,BITE) EXTERNAL;
 29   2   =               DECLARE AFT ADDRESS, (LENGTH,BITE) BYTE;
 30   2   =             END BINARY;
 31   1   =     HEX:    PROCEDURE(AFT,LENGTH,BUFPTR) EXTERNAL;
 32   2   =               DECLARE(AFT,LENGTH,BUFPTR) ADDRESS;
 33   2   =             END HEX;
 34   1   =     INTEG:  PROCEDURE(AFT,BITE) EXTERNAL;
 35   2   =               DECLARE AFT ADDRESS, BITE BYTE;
 36   2   =             END INTEG;
 37   1   =     INTEGD: PROCEDURE(AFT,TWOBITES) EXTERNAL;
 38   2   =               DECLARE(AFT,TWOBITES) ADDRESS;
 39   2   =             END INTEGD;
 40   1   =     PAGE:   PROCEDURE(AFT) EXTERNAL;
 41   2   =               DECLARE AFT ADDRESS;
 42   2   =             END PAGE;
 43   1   =     PPAGE:  PROCEDURE(AFT) EXTERNAL;
 44   2   =               DECLARE AFT ADDRESS;
 45   2   =             END PPAGE;
 46   1   =     SKIP:   PROCEDURE(AFT) EXTERNAL;
 47   2   =               DECLARE(AFT) ADDRESS;
```

```
 48   2  =            END SKIP;
 49   1  =    SPACE:  PROCEDURE(AFT,LENGTH) EXTERNAL;
 50   2  =              DECLARE(AFT,LENGTH) ADDRESS;
 51   2  =            END SPACE;
         =
 52   1  =    EQUAL:  PROCEDURE(P1,P2,L) BYTE EXTERNAL;
 53   2  =              DECLARE(P1,P2) ADDRESS, L BYTE;
 54   2  =            END EQUAL;
 55   1  =    SETBIT: PROCEDURE(P,I,L,V) EXTERNAL;
 56   2  =              DECLARE P ADDRESS, (I,L,V) BYTE;
 57   2  =            END SETBIT;
 58   1  =    EQUBIT: PROCEDURE(P,I,L,V) BYTE EXTERNAL;
 59   2  =              DECLARE P ADDRESS, (I,L,V) BYTE;
 60   2  =            END EQUBIT;
```

```
                $EJECT
                /*****************************************************************/
                /*                                                               */
                /*      DECLARATION OF EXTERNAL VARIABLES AND PROCEDURES         */
                /*                                                               */
                /*****************************************************************/

 61   1             DECLARE (SI,BO,LP,KB,CRT) ADDRESS EXTERNAL;
 62   1             DECLARE (CR,LF,TAB) BYTE EXTERNAL;
 63   1             DECLARE E BYTE EXTERNAL;
 64   1             DECLARE ENDSYMBTAB ADDRESS EXTERNAL;
 65   1             DECLARE FIRSTAVAILADDR ADDRESS EXTERNAL;
 66   1             DECLARE PAGENO ADDRESS EXTERNAL;
 67   1             DECLARE LINENO BYTE EXTERNAL;
 68   1             HEADER:PROCEDURE(SW) EXTERNAL;
 69   2                    DECLARE SW BYTE;
 70   2                    END HEADER;
 71   1             EOL:   PROCEDURE EXTERNAL;
 72   2                    END EOL;
```

```
                $EJECT

                        /*****************************************/
                        /*                                       */
                        /*    LABEL DECODING AND ADDRESS         */
                        /*          ASSIGNMENT BLOCK             */
                        /*                                       */
                        /*****************************************/

 73   1     PASS1:  PROCEDURE PUBLIC;
 74   2             DECLARE BUFF(256) BYTE;      /* 1 BLOCK AS READ FROM SI */
 75   2             DECLARE BUFFPTR BYTE;        /* POINTER TO BUFF */
 76   2             DECLARE CODE(256) BYTE;      /* 1 MICROWORD W/O BLANKS */
 77   2             DECLARE CODEPTR BYTE;        /* POINTER TO CODE */
 78   2             DECLARE EOF BYTE;
 79   2             DECLARE LABL(8) BYTE;       /* ALPHANUMERIC LABEL */
 80   2             DECLARE SUFFIX BYTE;        /* BINARY LABEL SUFFIX */
 81   2             DECLARE SUFFIXLENGTH BYTE; /* NUMBER OF BITS IN SUFFIX */
 82   2             DECLARE SYMBTABPTR ADDRESS;
 83   2             DECLARE STATUS ADDRESS;

 84   2             BUFFADV:PROCEDURE;
 85   3                     DECLARE (NCHAR,STATUS) ADDRESS;
 86   3                     BUFFPTR = BUFFPTR + 1;
 87   3                     IF BUFFPTR=0        /* I.E. =256 */
                               THEN DO;
 89   4                          CALL READ(SI,.BUFF,256,.NCHAR,.STATUS);
 90   4                          IF NCHAR=0
                                   THEN DO;
 92   5                               CALL EOL;
 93   5                               CALL ALPHA(LP,36,.('*** MISSING',
                                       ' [END]. ASSEMBLY ABORTED.'));
 94   5                               CALL EOL;
 95   5                               CALL PAGE(LP);
 96   5                               CALL PAGE(LP);
 97   5                               CALL EXIT;
 98   5                               END;
 99   4                          BUFFPTR = 0;
100   4                          END;
101   3                   END BUFFADV;
```

```
                $EJECT

                        /* EXTRACTS AND COMPRESSES CODE FOR 1 MICROWORD      */
                        /* SETS E0F FLAG IF A ']'-CHARACTER IS ENCOUNTERED.  */
                        /* RESETS CODEPTR.                                   */

102   2                  NEWWORD:PROCEDURE;
103   3                          DECLARE (I,CHAR) BYTE;
104   3                          CODEADV:PROCEDURE;
105   4                                  IF CODEPTR<255
                                            THEN CODEPTR = CODEPTR + 1;
107   4                                  END CODEADV;

108   3                          DO I = 0 TO 255;
109   4                          CODE(I) = ' ';
110   4                          END;

111   3                          CODEPTR = 0;

112   3                          CHAR = 00H;
113   3                          DO WHILE CHAR<>';';
114   4                          CHAR = BUFF(BUFFPTR);
115   4                          CALL BUFFADV;

116   4                          IF CHAR=CR
                                    THEN DO;
118   5                                  IF BUFF(BUFFPTR)=LF THEN CALL BUFFADV;
120   5                                  END;

121   4                          IF CHAR='*' OR CHAR=';'
                                    THEN DO WHILE BUFF(BUFFPTR)<>CR;
123   5                                  CALL BUFFADV;
124   5                                  END;

125   4                          IF CHAR=';'
                                    THEN DO;
127   5                                  CALL BUFFADV; /* STEP OVER CR. */
128   5                                  IF BUFF(BUFFPTR)=LF THEN CALL BUFFADV;
130   5                                  CODE(CODEPTR) = ';';
131   5                                  CALL CODEADV;
132   5                                  END;

133   4                          IF CHAR<>CR AND CHAR<>TAB AND CHAR<>' '
                                          AND CHAR<>'*' AND CHAR<>';'
                                    THEN DO;
135   5                                  CODE(CODEPTR) = CHAR;
136   5                                  CALL CODEADV;
137   5                                  END;

138   4                          IF CHAR=']' THEN DO;
140   5                                           E0F = 0FFH;
141   5                                           RETURN;
142   5                                           END;
143   4                          END;

144   3                          CODEPTR = 0;
145   3                          END NEWWORD;
```

```
                $EJECT

                      /* EXTRACTS LABEL FROM MICROWORD IN ARRAY CODE */

146   2                 EXTRACTLABEL PROCEDURE;
147   3                         DECLARE (I,J,L) BYTE;
148   3                         CALL MOVE(8,.(' '),.LABL);
149   3                         L = 0;
150   3                         DO WHILE (CODE(L)<>':') AND (L<18);
151   4                         L = L + 1;
152   4                         END;
153   3                         IF (L=18) AND (CODE(17)<>':')
                                   THEN DO;
155   4                                SUFFIX = 0;
156   4                                SUFFIXLENGTH = 0;
157   4                                RETURN;
158   4                                END;
159   3                         IF (CODE(0)<'A') OR (CODE(0)>'Z')
                                   THEN DO;
161   4                                E = E + 1;
162   4                                CALL ALPHA(LP,16,.('*** LABEL FIELD '));
163   4                                CALL ALPHA(LP,L,.CODE);
164   4                                CALL ALPHA(LP,46,.(' DOES NOT BEGIN ',
                                       'WITH AN ALPHABETIC CHARACTER. '));
165   4                                CALL EOL;
166   4                                END;
167   3                         I = 0;
168   3                         DO WHILE (CODE(I)<>'(') AND (I<L) AND (I<8);
169   4                         IF ((CODE(I)<'A') OR (CODE(I)>'Z')) AND
                                   ((CODE(I)<'0') OR (CODE(I)>'9'))
                                    THEN DO;
171   5                                 E = E + 1;
172   5                                 CALL ALPHA(LP,16,.('*** LABEL FIELD '));
173   5                                 CODE(I) = '?';
174   5                                 CALL ALPHA(LP,L,.CODE);
175   5                                 CALL ALPHA(LP,37,.(' CONTAINS NON-',
                                        'ALPHANUMERIC CHARACTER. '));
176   5                                 CALL EOL;
177   5                                 END;
178   4                         LABL(I) = CODE(I);
179   4                         I = I + 1;
180   4                         END;
181   3                         IF CODE(I)=':' THEN DO;
183   4                                             SUFFIX = 0;
184   4                                             SUFFIXLENGTH = 0;
185   4                                             RETURN;
186   4                                             END;
187   3                         IF (CODE(I)<>'(') OR (CODE(L-1)<>')')
                                   THEN DO;
189   4                                E = E + 1;
190   4                                CALL ALPHA(LP,16,.('*** LABEL FIELD '));
191   4                                CALL ALPHA(LP,L,.CODE);
192   4                                CALL ALPHA(LP,42,.(' DOES NOT HAVE ',
                                       'PARANTHESIS WHERE EXPECTED. '));
193   4                                CALL EOL;
194   4                                SUFFIX = 0FFH;
195   4                                SUFFIXLENGTH = 0FFH;
```

```
196   4                          RETURN;
197   4                          END;
198   3                    I = I + 1;          /* STEP OVER (  */
199   3                    L = L - 2;          /* STEP OVER ): */
200   3                    DO J = I TO L;
201   4                    IF (CODE(J)<>'0') AND (CODE(J)<>'1')
                              THEN DO;
203   5                          CALL ALPHA(LP,16,.('*** LABEL FIELD '));
204   5                          CALL ALPHA(LP,L+2,.CODE);
205   5                          CALL ALPHA(LP,48,.(' CONTAINS CHARACTER ',
                                 'OTHER THAN 0 OR 1 IN SUFFIX.'));
206   5                          CALL EOL;
207   5                          SUFFIX = 0FFH;
208   5                          SUFFIXLENGTH = 0FFH;
209   5                          RETURN;
210   5                          END;
211   4                    END;
212   3                    SUFFIXLENGTH = L - I + 1;
213   3                    SUFFIX = 0;
214   3                    DO J = 0 TO SUFFIXLENGTH-1;
215   4                    IF CODE(I+J)='1'
                             THEN CALL SETBIT(.SUFFIX,8-SUFFIXLENGTH+J,1,1B);
217   4                    END;
218   3                    RETURN;
219   3                    END EXTRACTLABEL;
```

```
$EJECT

220   2             E = 0;
221   2             SYMBTABPTR = 0;
222   2             E0F = 0;
223   2             BUFFPTR = 255;
224   2             CALL BUFFADV;
```

```
                $EJECT

                        /*****************************/
                        /*   BUILD SYMBOL TABLE      */
                        /*****************************/

225   2                 CALL NEWWORD;
226   2                 DO WHILE NOT E0F;
227   3                 CALL EXTRACTLABEL;
228   3                 IF (.MEMORY+SYMBTABPTR) > 0FC56H
                          THEN DO;
230   4                       CALL ALPHA(LP,41,.('***** FATAL ERROR: SYMBOL ',
                                  'TABLE OVERFLOW.'));
231   4                       CALL EOL;
232   4                       CALL PAGE(LP);
233   4                       CALL PAGE(LP);
234   4                       CALL EXIT;
235   4                       END;
236   3                 CALL MOVE(8,.LABL,.MEMORY+SYMBTABPTR);
237   3                 MEMORY(SYMBTABPTR+8) = SUFFIXLENGTH;
238   3                 MEMORY(SYMBTABPTR+9) = SUFFIX;
239   3                 SYMBTABPTR = SYMBTABPTR + 12;
240   3                 CALL NEWWORD;
241   3                 END;

242   2                 ENDSYMBTAB = SYMBTABPTR - 1;

                        /*************************************/
                        /*   ABORT IF ANY ERRORS SO FAR      */
                        /*************************************/

243   2                 IF E > 0
                          THEN DO;
245   3                       CALL ALPHA(LP,47,.('***** ASSEMBLY ABORTED ',
                                  'BECAUSE OF LABEL ERRORS.'));
246   3                       CALL EOL;
247   3                       CALL PAGE(LP);
248   3                       CALL PAGE(LP);
249   3                       CALL EXIT;
250   3                       END;
```

```
$EJECT

                /*****************************************/
                /*   CHECK CORRECTNESS OF SYMBOL TABLE   */
                /*****************************************/

251   2         DO;
252   3         DECLARE (I,J) ADDRESS;
253   3         DECLARE (K,L,M,N) BYTE;
254   3         DECLARE BLSZERR BYTE;

255   3         I = 0;
256   3         BLSZERR = 0;
257   3         DO WHILE I < ENDSYMBTAB;
258   4         M = MEMORY(I+8);                    /* SUFFIX LENGTH */
259   4         IF (M<>0) AND (M<>1) AND (M<>3) AND (M<>4) AND (M<>7)
                   THEN DO;
261   5                   E = E + 1;
262   5                   BLSZERR = BLSZERR + 1;
263   5                   CALL ALPHA(LP,66,.('*** ILLEGAL SUFFIXLENGTH ',
                          'OR ILLEGAL SUFFIX USE IN BLOCK LABELLED: '));
264   5                   CALL ALPHA(LP,8,.MEMORY+I);
265   5                   CALL EOL;
266   5                   END;
267   4         IF M > 0
                   THEN L = SHL(01H,M);    /* # OF WORDS IN BLOCK */
269   4                ELSE L = 01H;
270   4         J = I + 12;
271   4         IF NOT EQUAL(.MEMORY(I),.('          '),8)
                   THEN DO WHILE EQUAL(.MEMORY(I),.MEMORY(J),8);
273   5                   J = J + 12;
274   5                   END;
275   4         IF (J-I)/12 <> L
                   THEN DO;
277   5                   BLSZERR = BLSZERR + 1;
278   5                   E = E + 1;
279   5                   CALL ALPHA(LP,19,.('*** BLOCK LABELLED '));
280   5                   CALL ALPHA(LP,8,.MEMORY+I);
281   5                   CALL ALPHA(LP,26,.(' IS TOO SHORT OR TOO',
                                              ' LONG. '));
282   5                   CALL EOL;
283   5                   END;
284   4                ELSE DO K = 0 TO (L-1);
285   5                   IF (MEMORY(I+12*K+8) <> MEMORY(I+8)) OR
                              (MEMORY(I+12*K+9) <> K)
                              THEN DO;
287   6                              BLSZERR = BLSZERR + 1;
288   6                              E = E + 1;
289   6                              CALL ALPHA(LP,41,.('*** ILLEGAL ',
                                     'SUFFIX USE IN BLOCK LABELLED '));
290   6                              CALL ALPHA(LP,8,.MEMORY+I);
291   6                              CALL EOL;
292   6                              END;
293   5                   END;
294   4         I = J;
295   4         END;
```

```
                $EJECT

                        /*************************************/
                        /*   ABORT IF ANY ERRORS SO FAR      */
                        /*************************************/

296   3                 IF BLSZERR > 0
                           THEN DO;
298   4                       CALL ALPHA(LP,52,.('*** ASSEMBLY ABORTED ',
                                  'BECAUSE OF SUFFIX USE ERRORS. '));
299   4                       CALL EOL;
300   4                       CALL PAGE(LP);
301   4                       CALL PAGE(LP);
302   4                       CALL EXIT;
303   4                       END;
304   3                 END;
```

```
$EJECT

                /*******************************/
                /*   COMPRESS SYMBOL TABLE     */
                /*******************************/

305   2         DO;
306   3         DECLARE (I,J) ADDRESS;
307   3         DECLARE K BYTE;

308   3         I,J = 0;
309   3         DO WHILE J < ENDSYMBTAB;
310   4         CALL MOVE(12,.MEMORY+J,.MEMORY(I));
311   4         IF MEMORY(J+8) = 0
                   THEN K = 01H;
313   4              ELSE K = SHL(01H,MEMORY(J+8));
314   4         J = J + K * 12;
315   4         I = I + 12;
316   4         END;
317   3         ENDSYMBTAB = I - 12;

318   3         END;
```

```
                $EJECT

                        /************************/
                        /*   ASSIGN ADDRESSES   */
                        /************************/

319   2                 DO;
320   3                 DECLARE ADDR ADDRESS;
321   3                 DECLARE I ADDRESS;

322   3                 ADDR = 0;
323   3                 I = 0;
324   3                 DO WHILE I <= ENDSYMBTAB;
325   4                 IF MEMORY(I+8) = 7
                          THEN DO;
327   5                       CALL MOVE(2,.ADDR,.MEMORY+I+10);
328   5                       ADDR = ADDR + 128;
329   5                       END;
330   4                 I = I + 12;
331   4                 END;
332   3                 I = 0;
333   3                 DO WHILE I <= ENDSYMBTAB;
334   4                 IF MEMORY(I+8) = 4
                          THEN DO;
336   5                       CALL MOVE(2,.ADDR,.MEMORY+I+10);
337   5                       ADDR = ADDR + 16;
338   5                       END;
339   4                 I = I + 12;
340   4                 END;
341   3                 I = 0;
342   3                 DO WHILE I <= ENDSYMBTAB;
343   4                 IF MEMORY(I+8) = 3
                          THEN DO;
345   5                       CALL MOVE(2,.ADDR,.MEMORY+I+10);
346   5                       ADDR = ADDR + 8;
347   5                       END;
348   4                 I = I + 12;
349   4                 END;
350   3                 I = 0;
351   3                 DO WHILE I <= ENDSYMBTAB;
352   4                 IF MEMORY(I+8) = 1
                          THEN DO;
354   5                       CALL MOVE(2,.ADDR,.MEMORY+I+10);
355   5                       ADDR = ADDR + 2;
356   5                       END;
357   4                 I = I + 12;
358   4                 END;
359   3                 I = 0;
360   3                 DO WHILE I <= ENDSYMBTAB;
361   4                 IF MEMORY(I+8) = 0
                          THEN DO;
363   5                       CALL MOVE(2,.ADDR,.MEMORY+I+10);
364   5                       ADDR = ADDR + 1;
365   5                       END;
366   4                 I = I + 12;
367   4                 END;
```

```
 368   3              FIRSTAVAILADDR = ADDR;
 369   3              END;
```

```
$EJECT

                /*****************************/
                /*   ALLOCATE SPACE ON DISK  */
                /*****************************/

370   2         DO;
371   3         DECLARE I ADDRESS;
372   3         DECLARE (K,M,N) BYTE;
373   3         DECLARE X(8) BYTE;

374   3         CALL MOVE(6,.(0,0,0,0,0,0),.X+2);

375   3         DO I = 0 TO ENDSYMBTAB BY 12;
376   4         K = MEMORY(I+8);
377   4         IF K > 0
                  THEN M = SHL(01H,K);
379   4         ELSE M = 01H;
380   4              DO N = 0 TO M-1;
381   5              X(0) = MEMORY(I+10) OR N;
382   5              X(1) = MEMORY(I+11);
383   5              CALL ALPHA(80,8,.X);
384   5              END;
385   4         END;

386   3         END;
```

```
$EJECT

                        /*******************************/
                        /*   SYMBOL TABLE LISTING      */
                        /*******************************/

387   2                 DO;
388   3                 DECLARE (I1, I2, I3) ADDRESS;
389   3                 DECLARE L BYTE;

390   3                 I1 = 0;
391   3                 I2 = 53 * 12;
392   3                 I3 = 2 * 53 * 12;
393   3                 L = 4;

394   3                 DO WHILE I1 <= ENDSYMBTAB;
395   4                 CALL SPACE(LP, 2);
396   4                 IF EQUAL(. ('          '), .MEMORY+I1, 8)
                            THEN CALL ALPHA(LP, 8, .('NO LABEL'));
398   4                       ELSE CALL ALPHA(LP, 8, .MEMORY+I1);
399   4                 CALL INTEGD(LP, MEMORY(I1+8));
400   4                 CALL SPACE(LP, 2);
401   4                 CALL HEX  (LP, 2, .MEMORY+I1+11);
402   4                 CALL HEX  (LP, 2, .MEMORY+I1+10);
403   4                 IF I2 <= ENDSYMBTAB
                            THEN DO;
405   5                       CALL SPACE(LP, 8);
406   5                       IF EQUAL(. ('          '), .MEMORY+I2, 8)
                                  THEN CALL ALPHA(LP, 8, .('NO LABEL'));
408   5                             ELSE CALL ALPHA(LP, 8, .MEMORY+I2);
409   5                       CALL INTEGD(LP, MEMORY(I2+8));
410   5                       CALL SPACE(LP, 2);
411   5                       CALL HEX  (LP, 2, .MEMORY+I2+11);
412   5                       CALL HEX  (LP, 2, .MEMORY+I2+10);
413   5                       END;
414   4                 IF I3 <= ENDSYMBTAB
                            THEN DO;
416   5                       CALL SPACE(LP, 8);
417   5                       IF EQUAL(. ('          '), .MEMORY+I3, 8)
                                  THEN CALL ALPHA(LP, 8, .('NO LABEL'));
419   5                             ELSE CALL ALPHA(LP, 8, .MEMORY+I3);
420   5                       CALL INTEGD(LP, MEMORY(I3+8));
421   5                       CALL SPACE(LP, 2);
422   5                       CALL HEX  (LP, 2, .MEMORY+I3+11);
423   5                       CALL HEX  (LP, 2, .MEMORY+I3+10);
424   5                       END;
425   4                 CALL EOL;
426   4                 L = L + 1;
427   4                 IF L = 57
                            THEN DO;
429   5                       L = 0;
430   5                       I1 = I3 + 12;
431   5                       I2 = I1 + 57 * 12;
432   5                       I3 = I2 + 57 * 12;
433   5                       END;
434   4                       ELSE DO;
435   5                       I1 = I1 + 12;
```

```
 436   5                    I2 = I2 + 12;
 437   5                    I3 = I3 + 12;
 438   5                    END;
 439   4               END;
 440   3               IF L>0 THEN CALL HEADER(0);

 442   3               END;
 443   2               CALL SEEK(SI,2,.(0,0),.(0,0),.STATUS);
 444   2               CALL SEEK(BO,2,.(0,0),.(0,0),.STATUS);
 445   2               END PASS1;
 446   1               END MICRO1;


MODULE INFORMATION:

     CODE AREA SIZE     = 0EC1H   3777D
     VARIABLE AREA SIZE = 0240H    576D
     MAXIMUM STACK SIZE = 000CH     12D
     558 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER    PASS II OF Q-80 MICROPROGRAM ASSEMBLER   27 OCT 77  PAGE   1


ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE MICRO2
OBJECT MODULE PLACED IN :F1:MICRO2.OBJ
COMPILER INVOKED BY:  PLM80 :F1:MICRO2.PLM DATE(27 OCT 77)


                $PRINT(:LP:) PAGEWIDTH(80)
                $TITLE('PASS II OF Q-80 MICROPROGRAM ASSEMBLER')
   1            MICRO2: DO;
                $INCLUDE(INCLUD.DCL)
   2    1   =   OPEN:    PROCEDURE(AFTPTR,FNPTR,ACCCODE,ECHOAFT,STATUS) EXTERNAL;
   3    2   =               DECLARE(AFTPTR,FNPTR,ACCCODE,ECHOAFT,STATUS) ADDRESS;
   4    2   =             END OPEN;
   5    1   =   CLOSE:   PROCEDURE(AFT,STATUS) EXTERNAL;
   6    2   =               DECLARE(AFT,STATUS) ADDRESS;
   7    2   =             END CLOSE;
   8    1   =   READ:    PROCEDURE(AFT,BUFPTR,LENGTH,COUNTPTR,STATUS) EXTERNAL;
   9    2   =               DECLARE(AFT,BUFPTR,LENGTH,COUNTPTR,STATUS) ADDRESS;
  10    2   =             END READ;
  11    1   =   WRITE:   PROCEDURE(AFT,BUFPTR,LENGTH,STATUS) EXTERNAL;
  12    2   =               DECLARE(AFT,BUFPTR,LENGTH,STATUS) ADDRESS;
  13    2   =             END WRITE;
  14    1   =   SEEK:    PROCEDURE(AFT,OPER,BLOCKNOPTR,BYTENOPTR,STATUS) EXTERNAL;
  15    2   =               DECLARE(AFT,OPER,BLOCKNOPTR,BYTENOPTR,STATUS) ADDRESS;
  16    2   =             END SEEK;
  17    1   =   DELETE:  PROCEDURE(AFT,STATUS) EXTERNAL;
  18    2   =                 DECLARE(AFT,STATUS) ADDRESS;
  19    2   =               END DELETE;
  20    1   =   ERROR:   PROCEDURE(ERRNO,STATUS) EXTERNAL;
  21    2   =               DECLARE(ERRNO,STATUS) ADDRESS;
  22    2   =             END ERROR;
  23    1   =   EXIT:    PROCEDURE EXTERNAL;
  24    2   =             END EXIT;
        =
  25    1   =   ALPHA:   PROCEDURE(AFT,LENGTH,BUFPTR) EXTERNAL;
  26    2   =               DECLARE(AFT,LENGTH,BUFPTR) ADDRESS;
  27    2   =             END ALPHA;
  28    1   =   BINARY:  PROCEDURE(AFT,LENGTH,BITE) EXTERNAL;
  29    2   =               DECLARE AFT ADDRESS, (LENGTH,BITE) BYTE;
  30    2   =             END BINARY;
  31    1   =   HEX:     PROCEDURE(AFT,LENGTH,BUFPTR) EXTERNAL;
  32    2   =               DECLARE(AFT,LENGTH,BUFPTR) ADDRESS;
  33    2   =             END HEX;
  34    1   =   INTEG:   PROCEDURE(AFT,BITE) EXTERNAL;
  35    2   =               DECLARE AFT ADDRESS, BITE BYTE;
  36    2   =             END INTEG;
  37    1   =   INTEGD:  PROCEDURE(AFT,TWOBITES) EXTERNAL;
  38    2   =               DECLARE(AFT,TWOBITES) ADDRESS;
  39    2   =             END INTEGD;
  40    1   =   PAGE:    PROCEDURE(AFT) EXTERNAL;
  41    2   =               DECLARE AFT ADDRESS;
  42    2   =             END PAGE;
  43    1   =   PPAGE:   PROCEDURE(AFT) EXTERNAL;
  44    2   =               DECLARE AFT ADDRESS;
  45    2   =             END PPAGE;
  46    1   =   SKIP:    PROCEDURE(AFT) EXTERNAL;
  47    2   =               DECLARE(AFT) ADDRESS;
```

```
 48   2  =            END SKIP;
 49   1  =   SPACE:   PROCEDURE(AFT,LENGTH) EXTERNAL;
 50   2  =              DECLARE(AFT,LENGTH) ADDRESS;
 51   2  =            END SPACE;
           =
 52   1  =   EQUAL:   PROCEDURE(P1,P2,L) BYTE EXTERNAL;
 53   2  =              DECLARE(P1,P2) ADDRESS, L BYTE;
 54   2  =            END EQUAL;
 55   1  =   SETBIT:  PROCEDURE(P,I,L,V) EXTERNAL;
 56   2  =              DECLARE P ADDRESS, (I,L,V) BYTE;
 57   2  =            END SETBIT;
 58   1  =   EQUBIT:  PROCEDURE(P,I,L,V) BYTE EXTERNAL;
 59   2  =              DECLARE P ADDRESS, (I,L,V) BYTE;
 60   2  =            END EQUBIT;
```

```
                $EJECT
                /*****************************************************************/
                /*                                                               */
                /*     DECLARATION OF EXTERNAL VARIABLES AND PROCEDURES          */
                /*                                                               */
                /*****************************************************************/

 61   1             DECLARE (SI, BO, LP, KB, CRT) ADDRESS EXTERNAL;
 62   1             DECLARE (CR, LF, TAB) BYTE EXTERNAL;
 63   1             DECLARE E BYTE EXTERNAL;
 64   1             DECLARE ENDSYMBTAB ADDRESS EXTERNAL;
 65   1             DECLARE FIRSTAVAILADDR ADDRESS EXTERNAL;
 66   1             DECLARE PAGENO ADDRESS EXTERNAL;
 67   1             DECLARE LINENO BYTE EXTERNAL;
 68   1             HEADER:PROCEDURE(SW) EXTERNAL;
 69   2                    DECLARE SW BYTE;
 70   2                    END HEADER;
 71   1             EOL:   PROCEDURE EXTERNAL;
 72   2                    END EOL;
```

```
                $EJECT
                /*****************************************************************/
                /*                                                               */
                /*         TRANSLATION OF MICROCODE                              */
                /*                                                               */
                /*****************************************************************/

                      /**********DECLARATION OF VARIABLES**********/

 73   1       PASS2:  PROCEDURE PUBLIC;
 74   2               DECLARE BUFF  (256) BYTE;    /* 1 BLOCK AS READ FROM SI.  */
 75   2               DECLARE LINE  ( 80) BYTE;    /* 1 LINE FOR LP.  */
 76   2               DECLARE CODE  (256) BYTE;    /* 1 MICROWORD W/O BLANKS.  */
 77   2               DECLARE PHRASE( 33) BYTE;    /* 1 PHRASE */
 78   2               DECLARE ADDR ADDRESS;
 79   2               DECLARE (BUFFPTR,CODEPTR) ADDRESS, LINEPTR BYTE;
 80   2               DECLARE P1BDIPHRASEPRESENT BYTE;
 81   2               DECLARE P2BREGPHRASEPRESENT BYTE;
 82   2               DECLARE DOALCPHRASEPRESENT BYTE;
 83   2               DECLARE NOTAD1 BYTE;
 84   2               DECLARE I BYTE;
```

```
            $EJECT

 85   2             DECLARE PHVECT( 18) BYTE;    /* PHRASE PRESENCE FLAGS */
 86   2             DECLARE TYPE BYTE;           /* PHRASE TYPE */

            /*       THE FOLLOWING TABLE GIVES THE MEANING OF 'TYPE'
                     AND IS ALSO USED TO INTERPRET THE SUBSCRIPT FOR 'PHVECT'
                              0 = GOTO - PHRASE,
                              1 = REG(R) - PHRASE,
                              2 = P1B - PHRASE,
                              3 = P2B - PHRASE,
                              4 = DO - PHRASE,
                              5 = ALC - PHRASE,
                              6 = SHIFTER - PHRASE,
                              7 = MXL0OUT - PHRASE,
                              8 = MXL1OUT - PHRASE,
                              9 = MXH0OUT - PHRASE,
                             10 = MXH1OUT - PHRASE,
                             11 = CO - PHRASES,
                             12 = MFLAGS - PHRASE,
                             13 = COUT - PHRASE,
                             14 = NOLOAD - PHRASE,
                             15 = I/O - PHRASE,
                             16 = GUA - PHRASE,
                             17 = E - PHRASE,
                             18 = INVALID PHRASE,
                             19 = END OF MICROWORD,
                             20 = UNRECOGNIZED PHRASE
```

```
            $EJECT

 87   2             DECLARE FIELD ( 22) BYTE;   /* FIELD WRITTEN Y/N */

            /*      THE FOLLOWING TABLE IS USED TO INTERPRET THE
                    SUBSCRIPTS FOR THE ARRAY 'FIELD'.
                              0 = RS
                              1 =
                              2 = C1
                              3 = C0
                              4 =
                              5 =
                              6 =
                              7 = A1
                              8 = A0
                              9 =
                             10 =
                             11 = S1
                             12 =
                             13 = R
                             14 = T
                             15 =
                             16 = M2
                             17 = M1
                             18 = M0
                             19 = D2
                             20 = D1
                             21 =
```

```
                $EJECT

 88   2                DECLARE MWORD ( 6) BYTE;   /* 1 MICROWORD */

                /*       MICROWORD FORMAT IS AS FOLLOWS:

                ---------------------------------------------------------
                !  R3  !  R2  !  R1  !  R0  !  RS  !  T2  !  T1  !  T0  !
                ---------------------------------------------------------
                ! DOE  !  M2  !  M1  !  M0  !  D2  !  D1  !  C1  !  C0  !
                ---------------------------------------------------------
                !  A1  !  A0  !  S1  !      !  IM  !  CO  !  F1  !  F0  !
                ---------------------------------------------------------
                ! SCY1 ! SCY0 ! CI1  ! CI0  ! I/O3 ! I/O2 ! I/O1 ! I/O0 !
                ---------------------------------------------------------
                ! GUA1 ! GUA0 !  E   ! FS3  ! FS2  ! FS1  ! FS0  ! ADDR9!
                ---------------------------------------------------------
                ! ADDR8! ADDR7! ADDR6! ADDR5! ADDR4! ADDR3! ADDR2! ADDR1!
                ---------------------------------------------------------  */
```

```
$EJECT

 89   2          DECLARE TBL(256) BYTE DATA(

            /*    ADDRESS BITS ARE INTERPRETED AS FOLLOWS:
                  BIT 7 = P2B - REG(T) - PHRASE PRESENT,
                  BIT 6 = A - BITS NOT EQUAL TO 01B,
                  BIT 5 = S1 SET BY A PHRASE,
                  BIT 4 = VALUE OF S1,
                  BIT 3 = A1 SET BY A PHRASE,
                  BIT 2 = A0 SET BY A PHRASE,
                  BIT 1 = VALUE OF A1,
                  BIT 0 = VALUE OF A0.

                  BITS OF EACH TABLE ENTRY ARE INTERPRETED AS FOLLOWS:

                  BIT 7 = UNFORSEEN CIRCUMSTANCE,
                  BIT 6 = NOT USED,
                  BIT 5 = NOT USED,
                  BIT 4 = NOT USED,
                  BIT 3 = CONFLICT CAN NOT BE RESOLVED,
                  BIT 2 = VALUE OF S1,
                  BIT 1 = VALUE OF A1,
                  BIT 0 = VALUE OF A0.                              */

00H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 00H, 80H, 80H, 80H, 00H, 05H, 02H, 03H,
80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H,
00H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 00H, 80H, 80H, 80H, 00H, 01H, 02H, 03H,
05H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 05H, 80H, 80H, 80H, 04H, 05H, 06H, 07H,
00H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 00H, 80H, 80H, 80H, 00H, 0DH, 02H, 03H,
80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H,
00H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 00H, 80H, 80H, 80H, 00H, 09H, 02H, 03H,
04H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 04H, 80H, 80H, 80H, 04H, 0DH, 06H, 07H,
00H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 00H, 80H, 80H, 80H, 00H, 05H, 02H, 03H,
80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H,
00H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 00H, 80H, 80H, 80H, 00H, 0DH, 02H, 03H,
05H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 05H, 80H, 80H, 80H, 0CH, 05H, 0EH, 0FH,
00H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 00H, 80H, 80H, 80H, 00H, 0DH, 02H, 03H,
80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 80H,
00H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 00H, 80H, 80H, 80H, 00H, 09H, 02H, 03H,
05H, 80H, 80H, 80H, 80H, 80H, 80H, 80H, 0DH, 80H, 80H, 80H, 0CH, 05H, 0EH, 0FH);
```

```
                $EJECT

                      /**********DECLARATION OF SUBROUTINES***********/

                      /* SUBROUTINE ADVANCES BUFFPTR 1 BYTE.  */
                      /* REFILLS BUFF IF REQUIRED.            */

 90   2                  BUFFADV:PROCEDURE;
 91   3                          DECLARE (NCHAR,STATUS) ADDRESS;
 92   3                          BUFFPTR = BUFFPTR + 1;
 93   3                          IF BUFFPTR=256
                                    THEN DO;
 95   4                                  CALL READ(SI,.BUFF,256,.NCHAR,.STATUS);
 96   4                                  IF NCHAR=0 THEN DO;
 98   5                                                  CALL EOL;
 99   5                                                  CALL EXIT;
100   5                                                  END;
101   4                                  BUFFPTR = 0;
102   4                                  END;
103   3                          END BUFFADV;
```

```
$EJECT

                /* EXTRACTS, PRINTS AND COMPRESSES CODE FOR 1 MICROWORD  */
                /* TERMINATES PROGRAM IF A ']'-CHARACTER IS ENCOUNTERED. */
                /* RESETS CODEPTR, PHVECT, AND FIELD.                    */
                /* INITIALIZES MWORD TO THE VALUE CORRESPONDING TO THE   */
                /* NULL MICROINSTRUCTION.                                */
                /* RESETS THE DELAYED RESOLUTION FLAGS.                  */
                /* INITIALIZES PHRASE(0) TO A VALUE OTHER THAN ';'.      */

104   2          NEWWORD:PROCEDURE;
105   3                  DECLARE (I,CHAR) BYTE;
106   3                  DECLARE (L,ST) ADDRESS;
107   3                  DECLARE NEXTADDR ADDRESS;

108   3                  LINEOUT:PROCEDURE;
109   4                        IF LINEPTR>0
                                  THEN CALL ALPHA(LP,LINEPTR,.LINE);
111   4                        CALL EOL;
112   4                        DO I = 0 TO 79;
113   5                        LINE(I) = ' ';
114   5                        END;
115   4                        LINEPTR = 0;
116   4                        END LINEOUT;

117   3                  LINEADV:PROCEDURE;
118   4                        LINEPTR = LINEPTR + 1;
119   4                        IF LINEPTR=80 THEN CALL LINEOUT;
121   4                        END LINEADV;

122   3                  CODEADV:PROCEDURE;
123   4                        CODEPTR = CODEPTR + 1;
124   4                        IF CODEPTR=256
                                  THEN DO;
126   5                             CALL SKIP(LP);
127   5                             CALL SKIP(CRT);
128   5                             E = E + 1;
129   5                             CALL ALPHA(LP ,33,.('MORE THAN ',
                                        '256 BYTES IN ',
                                        'MICROWORD.'));
130   5                             CALL ALPHA(CRT,33,.('MORE THAN ',
                                        '256 BYTES IN ',
                                        'MICROWORD.'));
131   5                             CALL SKIP(CRT);
132   5                             CALL EXIT;
133   5                             END;
134   4                        END CODEADV;

135   3                  DO I = 0 TO 255;
136   4                  CODE(I) = ' ';
137   4                  END;

138   3                  CODEPTR = 0;

139   3                  DO I = 0 TO 79;
140   4                  LINE(I) = ' ';
141   4                  END;
```

```
142   3                    LINEPTR = 0;

143   3                    CHAR = 00H;
144   3                    DO WHILE CHAR<>';';
145   4                    CHAR = BUFF(BUFFPTR);
146   4                    CALL BUFFADV;

147   4                    IF CHAR=CR
                              THEN DO;
149   5                         CALL LINEOUT;
150   5                         IF BUFF(BUFFPTR)=LF THEN CALL BUFFADV;
152   5                         END;

153   4                    IF CHAR=TAB
                              THEN DO;
155   5                         CALL LINEADV;
156   5                                 DO WHILE (LINEPTR MOD 8)<>0;
157   6                                 CALL LINEADV;
158   6                                 END;
159   5                         END;

160   4                    IF CHAR=' ' THEN CALL LINEADV;

162   4                    IF CHAR='*' OR CHAR=';'
                              THEN DO;
164   5                         LINE(LINEPTR) = CHAR;
165   5                         CALL LINEADV;
166   5                             DO WHILE BUFF(BUFFPTR)<>CR;
167   6                             LINE(LINEPTR) = BUFF(BUFFPTR);
168   6                             CALL LINEADV;
169   6                             CALL BUFFADV;
170   6                             IF LINE(LINEPTR-1)=TAB
                                        THEN DO;
172   7                                     LINE(LINEPTR-1) = ' ';
173   7                                       DO WHILE (LINEPTR MOD 8)<>0;
174   8                                       CALL LINEADV;
175   8                                       END;
176   7                                     END;
177   6                             END;
178   5                         END;

179   4                    IF CHAR=';'
                              THEN DO;
181   5                         CALL LINEOUT;
182   5                         CALL BUFFADV; /* STEP OVER CR. */
183   5                         IF BUFF(BUFFPTR)=LF THEN CALL BUFFADV;
185   5                         CODE(CODEPTR) = ';';
186   5                         CALL CODEADV;
187   5                         END;

188   4                    IF CHAR<>CR AND CHAR<>TAB AND CHAR<>' '
                                   AND CHAR<>'*' AND CHAR<>';'
                              THEN DO;
190   5                         LINE(LINEPTR),CODE(CODEPTR) = CHAR;
191   5                         CALL LINEADV;
192   5                         CALL CODEADV;
```

```
193   5                         END;

194   4                   IF CHAR=']' THEN DO;
196   5                                    CALL EOL;
197   5                                    CALL EOL;
198   5                                    I = HIGH(FIRSTAVAILADDR);
199   5                                    CALL HEX(LP,2,.I);
200   5                                    I = LOW(FIRSTAVAILADDR);
201   5                                    CALL HEX(LP,2,.I);
202   5                                    CALL ALPHA(LP,27,.(' IS NEXT ',
                                               'AVAILABLE ADDRESS.'));
203   5                                    CALL EOL;
204   5                                    CALL INTEG(LP,E);
205   5                                    CALL INTEG(CRT,E);
206   5                                    CALL ALPHA(LP,6,.(' ERROR'));
207   5                                    CALL ALPHA(CRT,6,.(' ERROR'));
208   5                                    IF E<>1
                                              THEN DO;
210   6                                              CALL ALPHA(LP,1,.('S'));
211   6                                              CALL ALPHA(CRT,1,.('S'));
212   6                                              END;
213   5                                    CALL PAGE(LP);
214   5                                    CALL PAGE(LP);
215   5                                    CALL SKIP(CRT);
216   5                                    CALL EXIT;
217   5                                    END;
218   4                   END;

219   3                   CODEPTR = 0;
220   3                   DO I = 0 TO 17;
221   4                   PHVECT(I) = 0;
222   4                   END;
223   3                   DO I = 0 TO 21;
224   4                   FIELD(I) = 0;
225   4                   END;
226   3                   MWORD(0) = 00H;
227   3                   MWORD(1) = 5CH;
228   3                   MWORD(2) = 00H;
229   3                   MWORD(3) = 00H;
230   3                   MWORD(4) = 0E0H;
231   3                   MWORD(5) = 00H;
232   3                   CALL READ(BO,.ADDR,2,.L,.ST);
233   3                   NEXTADDR = ADDR + 1;
234   3                   IF (NEXTADDR AND 0001H) = 0001H
                              THEN CALL SETBIT(.MWORD,35,4,0001B);
236   3                         ELSE CALL SETBIT(.MWORD,35,4,0000B);
237   3                   NEXTADDR = SHR(NEXTADDR,1);
238   3                   MWORD(5) = LOW(NEXTADDR);
239   3                   IF (NEXTADDR AND 0100H) = 0100H
                              THEN CALL SETBIT(.MWORD,39,1,1B);
241   3                         ELSE CALL SETBIT(.MWORD,39,1,0B);
242   3                   P1BDIPHRASEPRESENT = 00H;
243   3                   P2BREGPHRASEPRESENT = 00H;
244   3                   NOTAD1 = 00H;
245   3                   DOALCPHRASEPRESENT = 00H;
246   3                   PHRASE(0) = '9';
```

```
247   3                    END NEWWORD;
```

```
                $EJECT

                /* EXTRACTS NEXT PHRASE FROM MICROWORD IN ARRAY CODE */

 248   2        NEWPHRASE:PROCEDURE;
 249   3                DECLARE I BYTE;
 250   3                IF CODE(CODEPTR) = ';'
                             THEN DO;          /* END OF MICROWORD */
 252   4                       PHRASE(0) = ';';
 253   4                       RETURN;
 254   4                       END;

 255   3                CALL MOVE(33,.('                                ',
                                  '),.PHRASE);

                        /* MOVE CODEPTR OVER ',' */
 256   3                  IF CODE(CODEPTR)=',' THEN CODEPTR = CODEPTR + 1;

                        /* TRANSFER NEXT PHRASE TO ARRAY PHRASE */
 258   3                  I = 0;
 259   3                  DO WHILE CODE(CODEPTR)<>';'
                               AND CODE(CODEPTR)<>','
                               AND I<33;
 260   4                  IF CODE(CODEPTR) = ':'
                             THEN DO;
 262   5                       I = 0;
 263   5                       CODEPTR = CODEPTR + 1;
 264   5                       CALL MOVE(33,.('                                ',
                                          '),.PHRASE);
 265   5                       END;
 266   4                  ELSE DO;
 267   5                       PHRASE(I) = CODE(CODEPTR);
 268   5                       CODEPTR = CODEPTR + 1;
 269   5                       I       = I       + 1;
 270   5                       END;
 271   4                  END;

                        /* IS PHRASE LONGER THAN 33 CHARACTERS ? */
 272   3                  IF I=33
                             THEN DO;
 274   4                       E = E + 1;
 275   4                       CALL ALPHA(LP,29,.('*** PHRASE IGNORED,',
                                        ' TOO LONG: '));
 276   4                       CALL ALPHA(LP,33,.PHRASE);
 277   4                       CALL ALPHA(LP,5,.('.....'));
 278   4                       CALL EOL;
 279   4                       PHRASE(0) = ' ';
 280   4                               DO WHILE CODE(CODEPTR)<>';'
                                             AND CODE(CODEPTR)<>',';
 281   5                               CODEPTR = CODEPTR + 1;
 282   5                               END;
 283   4                       END;
 284   3                  END NEWPHRASE;
```

```
                $EJECT

                     /* THIS SUBROUTINE, GIVEN THE ARRAY PHRASE, DETERMINES    */
                     /* THE TYPE OF THE PHRASE.  IT THEN CHECKS THE ARRAY      */
                     /* PHVECT TO SEE IF THIS TYPE OF PHRASE HAS OCCURED YET   */
                     /* IN THE CURRENT MICROINSTRUCTION.  IF NOT, THEN IT SETS */
                     /* THE CORRESPONDING ENTRY IN PHVECT.                     */

285   2                IDENTIFY:PROCEDURE;
286   3                        IF EQUAL(.PHRASE,.('GOTO'),4)
                               THEN DO;
288   4                             TYPE = 0;
289   4                             GOTO L;
290   4                             END;
291   3                        IF EQUAL(.PHRASE   ,.('REG('),4) AND
                                 (EQUAL(.PHRASE(5),.(')='  '),2) OR
                                  EQUAL(.PHRASE(5),.(')+1='),4))
                                  THEN DO;
293   4                             TYPE = 1;
294   4                             GOTO L;
295   4                             END;
296   3                        IF EQUAL(.PHRASE,.('P1B='),4)
                               THEN DO;
298   4                             TYPE = 2;
299   4                             GOTO L;
300   4                             END;
301   3                        IF EQUAL(.PHRASE,.('P2B='),4)
                               THEN DO;
303   4                             TYPE = 3;
304   4                             GOTO L;
305   4                             END;
306   3                        IF EQUAL(.PHRASE,.('DO='),3)
                               THEN DO;
308   4                             TYPE = 4;
309   4                             GOTO L;
310   4                             END;
311   3                        IF EQUAL(.PHRASE,.('ALC='),4)
                               THEN DO;
313   4                             TYPE = 5;
314   4                             GOTO L;
315   4                             END;
316   3                        IF EQUAL(.PHRASE,.('SHIFTER='),8)
                               THEN DO;
318   4                             TYPE = 6;
319   4                             GOTO L;
320   4                             END;
321   3                        IF EQUAL(.PHRASE,.('MXL0OUT='),8)
                               THEN DO;
323   4                             TYPE = 7;
324   4                             GOTO L;
325   4                             END;
326   3                        IF EQUAL(.PHRASE,.('MXL1OUT='),8)
                               THEN DO;
328   4                             TYPE = 8;
329   4                             GOTO L;
330   4                             END;
331   3                        IF EQUAL(.PHRASE,.('MXH0OUT='),8)
```

```
                                    THEN DO;
333   4                                   TYPE = 9;
334   4                                   GOTO L;
335   4                                   END;
336   3                             IF EQUAL(.PHRASE,.('MXH1OUT='),8)
                                    THEN DO;
338   4                                   TYPE = 10;
339   4                                   GOTO L;
340   4                                   END;
341   3                             IF EQUAL(.PHRASE,.('CO='),3)
                                    THEN DO;
343   4                                   TYPE = 11;
344   4                                   GOTO L;
345   4                                   END;
346   3                             IF EQUAL(.PHRASE,.('MFLAGS='),7)
                                    THEN DO;
348   4                                   TYPE = 12;
349   4                                   GOTO L;
350   4                                   END;
351   3                             IF EQUAL(.PHRASE,.('COUT='),5)
                                    THEN DO;
353   4                                   TYPE = 13;
354   4                                   GOTO L;
355   4                                   END;
356   3                             IF EQUAL(.PHRASE,.('NOLOAD '),7)
                                    THEN DO;
358   4                                   TYPE = 14;
359   4                                   GOTO L;
360   4                                   END;
361   3                             IF EQUAL(.PHRASE,.('I/O='),4)
                                    THEN DO;
363   4                                   TYPE = 15;
364   4                                   GOTO L;
365   4                                   END;
366   3                             IF EQUAL(.PHRASE,.('GUA='),4)
                                    THEN DO;
368   4                                   TYPE = 16;
369   4                                   GOTO L;
370   4                                   END;
371   3                             IF EQUAL(.PHRASE,.('E='),2)
                                    THEN DO;
373   4                                   TYPE = 17;
374   4                                   GOTO L;
375   4                                   END;
376   3                             IF EQUAL(.PHRASE,.(' '),1)
                                    THEN DO;
378   4                                   TYPE = 18;
379   4                                   GOTO L;
380   4                                   END;
381   3                             IF EQUAL(.PHRASE,.(';'),1)
                                    THEN DO;
383   4                                   TYPE = 19;
384   4                                   GOTO L;
385   4                                   END;
                            /*      IF NONE OF THE ABOVE THEN       */
386   3                                   TYPE = 20;
387   3                           L:
```

```
$EJECT

                        /* CHECK AND SET PHRASE PRESENCE INDICATOR */

                     IF TYPE<18
                        THEN IF PHVECT(TYPE)
                                THEN DO;
390   4                              E = E + 1;
391   4                              CALL ALPHA(LP,30,.('*** DUPLICAT',
                                             'E PHRASE IGNORED: '));
392   4                              CALL ALPHA(LP,32,.PHRASE);
393   4                              CALL EOL;
394   4                              TYPE = 18;
395   4                              END;
396   3                     IF TYPE<18 THEN PHVECT(TYPE) = 0FFH;

398   3                     END IDENTIFY;
```

```
                $EJECT

 399   2              ASCII2BIN:PROCEDURE(DIGIT) ADDRESS;
 400   3                      DECLARE DIGIT BYTE;
 401   3                      DECLARE (VALUE,BAD) BYTE;
 402   3                      DECLARE DUMMY ADDRESS AT(.VALUE);
 403   3                      IF DIGIT>=30H AND DIGIT<=3FH OR
                                  DIGIT>=41H AND DIGIT<=46H
                                  THEN DO;
 405   4                                IF EQUBIT(.DIGIT,1,3,011B)
                                             THEN VALUE = DIGIT AND 0FH;
 407   4                                IF EQUBIT(.DIGIT,1,3,100B)
                                             THEN VALUE = (DIGIT AND 0FH) + 9;
 409   4                                BAD = 00H;
 410   4                                END;
 411   3                           ELSE DO;
 412   4                                VALUE = 00H;
 413   4                                BAD = 0FFH;
 414   4                                END;
 415   3                      RETURN DUMMY;
 416   3                      END ASCII2BIN;
```

```
                $EJECT

417   2             PRINTROUTINE:PROCEDURE;
418   3                     DECLARE Z BYTE;

                          /* WRITE INTO BINARY FILE */

419   3                     CALL ALPHA(80,6,.MWORD);

                          /* PRINT CONTENTS OF MICROWORD */

420   3                     CALL EOL;
421   3                     IF LINENO>53 THEN CALL HEADER(0);
423   3                     CALL ALPHA(LP,14,.('--------------'));
424   3                     CALL EOL;
425   3                     CALL ALPHA(LP,9,.('! ADDR = '));
426   3                     Z = SHL(HIGH(ADDR),4);
427   3                     CALL HEX  (LP,1,.Z);
428   3                     CALL HEX  (LP,2,.ADDR);
429   3                     CALL ALPHA(LP,2,.(' !'));
430   3                     CALL EOL;
431   3                     CALL ALPHA(LP,80,.('------------------------------',
                                                '------------------------------',
                                                '------------------------'));
432   3                     CALL EOL;
433   3                     CALL ALPHA(LP,6,.('! R = '));
434   3                     IF EQUBIT(.MWORD,4,1,1B)
                                THEN CALL ALPHA(LP,1,.('X'));
436   3                        ELSE CALL HEX  (LP,1,.MWORD);
437   3                     CALL ALPHA(LP,9,.(' ! M   = '));
438   3                     Z = MWORD(1) AND 70H;
439   3                     CALL HEX(LP,1,.Z);
440   3                     CALL ALPHA(LP,7,.(' ! D = '));
441   3                     Z = SHL(MWORD(1),3) AND 60H OR 10H;
442   3                     CALL HEX(LP,1,.Z);
443   3                     CALL ALPHA(LP,7,.(' ! A = '));
444   3                     Z = SHR(MWORD(2),2) AND 30H;
445   3                     CALL HEX(LP,1,.Z);
446   3                     CALL ALPHA(LP,10,.(' !! SCY = '));
447   3                     Z = SHR(MWORD(3),2) AND 30H;
448   3                     CALL HEX(LP,1,.Z);
449   3                     CALL ALPHA(LP,9,.(' ! CI  = '));
450   3                     Z = MWORD(3) AND 30H;
451   3                     CALL HEX(LP,1,.Z);
452   3                     CALL ALPHA(LP,9,.(' ! GUA = '));
453   3                     Z = SHR(MWORD(4),2) AND 30H;
454   3                     CALL HEX(LP,1,.Z);
455   3                     CALL ALPHA(LP,11,.(' !! ADDR = '));
456   3                     Z = (SHL(MWORD(4),5) AND 20H) OR
                               (SHR(MWORD(5),3) AND 10H);
457   3                     CALL HEX(LP,1,.Z);
458   3                     Z = SHL(MWORD(5),1) AND 0FEH;
459   3                     CALL HEX(LP,2,.Z);
460   3                     CALL ALPHA(LP,2,.(' !'));
461   3                     CALL EOL;
462   3                     CALL ALPHA(LP,80,.('------------------------------',
                                                '------------------------------',
```

```
                                                 '------------------------'));
463   3                          CALL EOL;
464   3                          CALL ALPHA(LP,6,.('! T = '));
465   3                          Z = SHL(MWORD(0),4) AND 70H;
466   3                          IF EQUBIT(.MWORD,4,1,1B)
                                    THEN CALL ALPHA(LP,1,.('X'));
468   3                                 ELSE CALL HEX  (LP,1,.Z);
469   3                          CALL ALPHA(LP,9,.(' ! DOE = '));
470   3                          Z = SHR(MWORD(1),3) AND 10H;
471   3                          CALL HEX(LP,1,.Z);
472   3                          CALL ALPHA(LP,7,.(' ! C = '));
473   3                          Z = SHL(MWORD(1),4) AND 30H;
474   3                          CALL HEX(LP,1,.Z);
475   3                          CALL ALPHA(LP,7,.(' ! S = '));
476   3                          IF EQUBIT(.MWORD,18,1,1B)
                                    THEN Z = 20H;
478   3                                 ELSE Z = 10H;
479   3                          CALL HEX(LP,1,.Z);
480   3                          CALL ALPHA(LP,10,.(' !! CO  = '));
481   3                          Z = SHL(MWORD(2),2) AND 10H;
482   3                          CALL HEX(LP,1,.Z);
483   3                          CALL ALPHA(LP,9,.(' ! I/O = '));
484   3                          Z = SHL(MWORD(3),4);
485   3                          CALL HEX(LP,1,.Z);
486   3                          CALL ALPHA(LP,9,.(' ! E   = '));
487   3                          Z = SHR(MWORD(4),1) AND 10H;
488   3                          CALL HEX(LP,1,.Z);
489   3                          CALL ALPHA(LP,11,.(' !! FLAG = '));
490   3                          Z = SHL(MWORD(4),3) AND 0F0H;
491   3                          CALL HEX(LP,1,.Z);
492   3                          CALL ALPHA(LP,4,.('   !'));
493   3                          CALL EOL;
494   3                          CALL ALPHA(LP,80,.('------------------------------',
                                                   '------------------------------',
                                                   '------------------------'));
495   3                          CALL EOL;
496   3                          CALL ALPHA(LP,2,.('! '));
497   3                          CALL HEX  (LP,12,.MWORD);
498   3                          CALL ALPHA(LP,2,.(' !'));
499   3                          CALL SPACE(LP,18);
500   3                          CALL ALPHA(LP,9,.('!! IM  = '));
501   3                          Z = SHL(MWORD(2),1) AND 10H;
502   3                          CALL HEX(LP,1,.Z);
503   3                          CALL ALPHA(LP,9,.(' ! F   = '));
504   3                          Z = SHL(MWORD(2),4) AND 30H;
505   3                          CALL HEX  (LP,1,.Z);
506   3                          CALL ALPHA(LP,9,.(' ! RS  = '));
507   3                          Z = SHL(MWORD(0),1) AND 10H;
508   3                          CALL HEX  (LP,1,.Z);
509   3                          CALL ALPHA(LP,3,.(' !!'));
510   3                          CALL EOL;
511   3                          CALL ALPHA(LP,16,.('----------------'));
512   3                          CALL SPACE(LP,18);
513   3                          CALL ALPHA(LP,33,.('---------------------',
                                                   '-------------'));
514   3                          CALL EOL;
515   3                          CALL EOL;
```

```
516   3                    END PRINTROUTINE;
```

```
$EJECT
```

```
            $EJECT

517   2             ADDR = 0;
518   2             E = 0;
519   2             BUFFPTR = 255;  /* INDICATES BUFF EMPTY */
520   2             CALL BUFFADV;   /* FILLS BUFF */


                  /**********CODE TRANSLATION LOOP**************/

521   2             DO WHILE 0FFH;  /* UNTIL EOF GETS US OUT */

                  /* ONE MICROWORD TRANSLATED EACH ITERATION */
                  /* INITIALIZE FOR NEXT MICROWORD */

522   3             CALL NEWWORD;

                  /* PROCESS ALL PHRASES OF THE MICROWORD */

523   3             DO WHILE PHRASE(0)<>';';
524   4             CALL NEWPHRASE;
525   4             CALL IDENTIFY;
```

```
                $EJECT

                        /* TRANSLATE PHRASE, CHECK FOR CONFLICTS */

526   4                 DO CASE TYPE;
                        /*******************/
                        /* 0: GOTO-PHRASE */
                        /*******************/
527   5                 DO;
528   6                 DECLARE (I,ADDR) ADDRESS;
529   6                 DECLARE LABL(8) BYTE;
530   6                 DECLARE (SUFFIX,SUFFIXLENGTH) BYTE;
531   6                 DECLARE (K,L) BYTE;

532   6                 IF PHRASE(4) = '<'
                          THEN DO;
534   7                       IF EQUAL(.('<000+@>'),.PHRASE(4),8)
                                THEN DO;
536   8                             CALL SETBIT(.MWORD,35,4,1111B);
537   8                             CALL SETBIT(.MWORD,39,1,0B);
538   8                             MWORD(5) = 0;
539   8                             GOTO END0;
540   8                             END;
541   7                       IF EQUAL(.('<256+@>'),.PHRASE(4),8)
                                THEN DO;
543   8                             CALL SETBIT(.MWORD,35,4,1111B);
544   8                             CALL SETBIT(.MWORD,39,1,0B);
545   8                             MWORD(5) = 80H;
546   8                             GOTO END0;
547   8                             END;
548   7                       IF EQUAL(.('<512+@>'),.PHRASE(4),8)
                                THEN DO;
550   8                             CALL SETBIT(.MWORD,35,4,1111B);
551   8                             CALL SETBIT(.MWORD,39,1,1B);
552   8                             MWORD(5) = 0;
553   8                             GOTO END0;
554   8                             END;
555   7                       IF EQUAL(.('<768+@>'),.PHRASE(4),8)
                                THEN DO;
557   8                             CALL SETBIT(.MWORD,35,4,1111B);
558   8                             CALL SETBIT(.MWORD,39,1,1B);
559   8                             MWORD(5) = 80H;
560   8                             GOTO END0;
561   8                             END;
562   7                       END;
563   6                 CALL MOVE(8,.('        '),.LABL);
564   6                 L = 0;
565   6                 DO WHILE (PHRASE(L+4)<>'(') AND
                                 (PHRASE(L+4)<>' ') AND
                                 (L<9);
566   7                 L = L + 1;
567   7                 END;
568   6                 IF (L=9) AND (PHRASE(8)<>'(')
                                 AND (PHRASE(8)<>' ')
                          THEN DO;
570   7                       E = E + 1;
571   7                       CALL ALPHA(LP,28,.('*** ILLEGAL LABEL  '),
```

```
                                                 'TOO LONG.  '));
572   7                    CALL EOL;
573   7                    GOTO END0;
574   7                    END;
575   6              IF (PHRASE(4)<'A') OR (PHRASE(4)>'Z')
                        THEN DO;
577   7                    E = E + 1;
578   7                    CALL ALPHA(LP,65, ('*** ILLEGAL LABEL.  ',
                           'DOES NOT BEGIN WITH AN ALPHABETIC ',
                           'CHARACTER.  '));
579   7                    CALL EOL;
580   7                    GOTO END0;
581   7                    END;
582   6              LABL(0) = PHRASE(4);
583   6              K = 5;
584   6              DO WHILE K < (L+4);
585   7              IF ((PHRASE(K)<'A') OR (PHRASE(K)>'Z')) AND
                        ((PHRASE(K)<'0') OR (PHRASE(K)>'9'))
                        THEN DO;
587   8                    E = E + 1;
588   8                    CALL ALPHA(LP,56, ('*** ILLEGAL LABEL.  ',
                           'CONTAINS NON-ALPHANUMERIC CHARACTER.  '));
589   8                    CALL EOL;
590   8                    GOTO END0;
591   8                    END;
592   7              LABL(K-4) = PHRASE(K);
593   7              END;
594   6              I = 0;
595   6              DO WHILE (I<=ENDSYMBTAB) AND
                           NOT EQUAL(.LABL,.MEMORY+I,8);
596   7              I = I + 12;
597   7              END;
598   6              IF I > ENDSYMBTAB
                        THEN DO;
600   7                    E = E + 1;
601   7                    CALL ALPHA(LP,34,.('*** NO SUCH LABEL IN ',
                                                 'SYMBOL TABLE.  '));
602   7                    CALL EOL;
603   7                    GOTO END0;
604   7                    END;
605   6              SUFFIXLENGTH = MEMORY(I+8);
606   6              CALL MOVE(2,.MEMORY+I+10,.ADDR);
607   6              IF (SUFFIXLENGTH=0) AND (PHRASE(L+4)<>' ')
                        THEN DO;
609   7                    E = E + 1;
610   7                    CALL ALPHA(LP,20,.('*** ILLEGAL SUFFIX.  '));
611   7                    CALL EOL;
612   7                    GOTO END0;
613   7                    END;
614   6              IF SUFFIXLENGTH = 0
                        THEN DO;
616   7                    IF (ADDR AND 0001H) = 0001H
                              THEN CALL SETBIT(.MWORD,35,4,0001B);
618   7                         ELSE CALL SETBIT(.MWORD,35,4,0000B);
619   7                    ADDR = SHR(ADDR,1);
620   7                    MWORD(5) = LOW(ADDR);
621   7                    IF (ADDR AND 0100H) = 0100H
```

```
                              THEN CALL SETBIT(.MWORD,39,1,1B);
623   7                       ELSE CALL SETBIT(.MWORD,39,1,0B);
624   7                  GOTO END0;
625   7                  END;
626   6            IF (PHRASE(L+4)<>'(') OR (PHRASE(L+SUFFIXLENGTH+5)<>')')
                      THEN DO;
628   7                  E = E + 1;
629   7                  CALL ALPHA(LP,20,.('*** ILLEGAL SUFFIX. '));
630   7                  CALL EOL;
631   7                  GOTO END0;
632   7                  END;
633   6            DO CASE SUFFIXLENGTH;
634   7      /* SUFFIXLENGTH = 0 */  ;
635   7      /* SUFFIXLENGTH = 1 */  DO;
636   8                                 L = L + 5;
637   8                                 IF PHRASE(L)='0'
                                          THEN CALL SETBIT(.MWORD,35,4,0000B);
639   8                                 IF PHRASE(L)='1'
                                          THEN CALL SETBIT(.MWORD,35,4,0001B);
641   8                                 IF PHRASE(L)='S'
                                          THEN CALL SETBIT(.MWORD,35,4,0010B);
643   8                                 IF PHRASE(L)='Z'
                                          THEN CALL SETBIT(.MWORD,35,4,0011B);
645   8                                 IF PHRASE(L)='P'
                                          THEN CALL SETBIT(.MWORD,35,4,0100B);
647   8                                 IF (PHRASE(L) <> '0') AND
                                          (PHRASE(L) <> '1') AND
                                          (PHRASE(L) <> 'S') AND
                                          (PHRASE(L) <> 'Z') AND
                                          (PHRASE(L) <> 'P')
                                          THEN DO;
649   9                                        E = E + 1;
650   9                                        CALL ALPHA(LP,20,.('*** ILLEGAL
                                                 'SUFFIX. '));
651   9                                        CALL EOL;
652   9                                        GOTO END0;
653   9                                        END;
654   8                                 ADDR = SHR(ADDR,1);
655   8                                 MWORD(5) = LOW(ADDR);
656   8                                 IF (ADDR AND 0100H) = 0100H
                                          THEN CALL SETBIT(.MWORD,39,1,1B);
658   8                                          ELSE CALL SETBIT(.MWORD,39,1,0B);
659   8                                 END;
660   7      /* SUFFIXLENGTH = 2 */  ;
661   7      /* SUFFIXLENGTH = 3 */  DO;
662   8                                 L = L + 5;
663   8                                 IF ((PHRASE(L  )='0') OR
                                              (PHRASE(L  )='1')) AND
                                          ((PHRASE(L+1)='0') OR
                                              (PHRASE(L+1)='1')) AND
                                          ((PHRASE(L+2)='0') OR
                                              (PHRASE(L+2)='1'))
                                          THEN DO;
665   9                                        SUFFIX = SHL(PHRASE(L  ) AND 1,2)
                                                     OR SHL(PHRASE(L+1) AND 1,1)
                                                     OR     (PHRASE(L+2) AND 1);
666   9                                        ADDR = ADDR OR SUFFIX;
```

```
 667   9                                        IF (ADDR AND 0001H) = 0001H
                                                   THEN CALL SETBIT(.MWORD,
                                                                    35,4,0001B);
 669   9                                           ELSE CALL SETBIT(.MWORD,
                                                                    35,4,0000B);
 670   9                                        ADDR = SHR(ADDR,1);
 671   9                                        MWORD(5) = LOW(ADDR);
 672   9                                        IF (ADDR AND 0100H) = 0100H
                                                   THEN CALL SETBIT(.MWORD,
                                                                    39,1,1B);
 674   9                                           ELSE CALL SETBIT(.MWORD,
                                                                    39,1,0B);
 675   9                                        GOTO END8WAY;
 676   9                                        END;
 677   8                                  IF EQUAL(.PHRASE+L,.('HHI'),3)
                                              THEN DO;
 679   9                                        CALL SETBIT(.MWORD,35,4,1001B);
 680   9                                        ADDR = SHR(ADDR,1);
 681   9                                        MWORD(5) = LOW(ADDR);
 682   9                                        IF (ADDR AND 0100H) = 0100H
                                                   THEN CALL SETBIT(.MWORD,
                                                                    39,1,1B);
 684   9                                           ELSE CALL SETBIT(.MWORD,
                                                                    39,1,0B);
 685   9                                        GOTO END8WAY;
 686   9                                        END;
                                        /* IF NONE OF THE ABOVE THEN   */
 687   8                                  E = E + 1;
 688   8                                  CALL ALPHA(LP,20,.('*** ILLEGAL ',
                                                              'SUFFIX. '));
 689   8                                  CALL EOL;
 690   8                          END8WAY: END;
 691   7       /* SUFFIXLENGTH = 4 */     DO;
 692   8                                  L = L + 5;
 693   8                                  IF ((PHRASE(L  )='0') OR
                                                (PHRASE(L  )='1')) AND
                                         ((PHRASE(L+1)='0') OR
                                                (PHRASE(L+1)='1')) AND
                                         ((PHRASE(L+2)='0') OR
                                                (PHRASE(L+2)='1')) AND
                                         ((PHRASE(L+3)='0') OR
                                                (PHRASE(L+3)='1'))
                                         THEN DO;
 695   9                                        SUFFIX = SHL(PHRASE(L  ) AND 1,3)
                                                       OR SHL(PHRASE(L+1) AND 1,2)
                                                       OR SHL(PHRASE(L+2) AND 1,1)
                                                       OR    (PHRASE(L+3) AND 1);
 696   9                                        ADDR = ADDR OR SUFFIX;
 697   9                                        IF (ADDR AND 0100H) = 0100H
                                                   THEN CALL SETBIT(.MWORD,
                                                                    35,4,0001B);
 699   9                                           ELSE CALL SETBIT(.MWORD,
                                                                    35,4,0000B);
 700   9                                        ADDR = SHR(ADDR,1);
 701   9                                        MWORD(5) = LOW(ADDR);
 702   9                                        IF (ADDR AND 0001H) = 0001H
                                                   THEN CALL SETBIT(.MWORD,
```

```
                                                                39,1,1B);
 704   9                                          ELSE CALL SETBIT(.MWORD,
                                                                39,1,0B);
 705   9                                       GOTO END16WAY;
 706   9                                       END;
 707   8                                 IF EQUAL(.PHRASE+L,.('RHHI'),4)
                                           THEN DO;
 709   9                                       CALL SETBIT(.MWORD,35,4,1000B);
 710   9                                       ADDR = SHR(ADDR,1);
 711   9                                       MWORD(5) = LOW(ADDR);
 712   9                                       IF (ADDR AND 0100H) = 0100H
                                                 THEN CALL SETBIT(.MWORD,
                                                                39,1,1B);
 714   9                                          ELSE CALL SETBIT(.MWORD,
                                                                39,1,0B);
 715   9                                       GOTO END16WAY;
 716   9                                       END;
                                       /* IF NONE OF THE ABOVE  */
 717   8                                 E = E + 1;
 718   8                                 CALL ALPHA(LP,19,.('*** ILLEGAL ',
                                                          'SUFFIX.  '));
 719   8                                 CALL EOL;
 720   8                           END16WAY: END;
 721   7      /* SUFFIXLENGTH = 5 */     ;
 722   7      /* SUFFIXLENGTH = 6 */     ;
 723   7      /* SUFFIXLENGTH = 7 */     DO;
 724   8                                 L = L + 5;
 725   8                                 IF ((PHRASE(L  )='0') OR
                                               (PHRASE(L  )='1')) AND
                                           ((PHRASE(L+1)='0') OR
                                               (PHRASE(L+1)='1')) AND
                                           ((PHRASE(L+2)='0') OR
                                               (PHRASE(L+2)='1')) AND
                                           ((PHRASE(L+3)='0') OR
                                               (PHRASE(L+3)='1')) AND
                                           ((PHRASE(L+4)='0') OR
                                               (PHRASE(L+4)='1')) AND
                                           ((PHRASE(L+5)='0') OR
                                               (PHRASE(L+5)='1')) AND
                                           ((PHRASE(L+6)='0') OR
                                               (PHRASE(L+6)='1'))
                                           THEN DO;
 727   9                                       SUFFIX = SHL(PHRASE(L  ) AND 1,6)
                                                     OR SHL(PHRASE(L+1) AND 1,5)
                                                     OR SHL(PHRASE(L+2) AND 1,4)
                                                     OR SHL(PHRASE(L+3) AND 1,3)
                                                     OR SHL(PHRASE(L+4) AND 1,2)
                                                     OR SHL(PHRASE(L+5) AND 1,1)
                                                     OR    (PHRASE(L+6) AND 1);
 728   9                                       ADDR = ADDR OR SUFFIX;
 729   9                                       IF (ADDR AND 0100H) = 0100H
                                                 THEN CALL SETBIT(.MWORD,
                                                                35,4,0001B);
 731   9                                          ELSE CALL SETBIT(.MWORD,
                                                                35,4,0000B);
 732   9                                       ADDR = SHR(ADDR,1);
 733   9                                       MWORD(5) = LOW(ADDR);
```

```
 734   9                                        IF (ADDR AND 0001H) = 0001H
                                                   THEN CALL SETBIT(.MWORD,
                                                                    39,1,1B);
 736   9                                           ELSE CALL SETBIT(.MWORD,
                                                                    39,1,0B);
 737   9                                        GOTO END128WAY;
 738   9                                        END;
 739   8                                  IF EQUAL(.PHRASE+L,.('EOPCODE'),7)
                                               THEN DO;
 741   9                                        CALL SETBIT(.MWORD,35,4,1100B);
 742   9                                        ADDR = SHR(ADDR,1);
 743   9                                        MWORD(5) = LOW(ADDR);
 744   9                                        IF (ADDR AND 0100H) = 0100H
                                                   THEN CALL SETBIT(.MWORD,
                                                                    39,1,1B);
 746   9                                           ELSE CALL SETBIT(.MWORD,
                                                                    39,1,0B);
 747   9                                        GOTO END128WAY;
 748   9                                        END;
                                        /* IF NONE OF THE ABOVE THEN  */
 749   8                                     E = E + 1;
 750   8                                     CALL ALPHA(LP,19,.('*** ILLEGAL ',
                                                              'SUFFIX: '));
 751   8                                     CALL EOL;
 752   8                           END128WAY: END;
 753   7                                  END;
 754   6       END0:    END;
```

```
                $EJECT

                /*********************/
                /* 1: REG(?)-PHRASE */
                /*********************/
 755    5               DO;
 756    6               DECLARE (REGNO,BADREG) BYTE;
 757    6               DECLARE REGDECODE ADDRESS AT(.REGNO);
 758    6               IF EQUAL(.('#'),.PHRASE(4),1)
                           THEN DO;
 760    7                      IF FIELD(0) AND EQUBIT(.MWORD,4,1,0B)
                                  THEN DO;
 762    8                            E = E +  ;
 763    8                            CALL ALP A(LP,37,.('*** CONFLICTING ',
                                     'SPECIFI ATION FOR RS.'));
 764    8                            CALL EOL
 765    8                            GOTO END
 766    8                            END;
 767    7                         ELSE DO;
 768    8                            REGNO  = 0;
 769    8                            BADREG = 0;
 770    8                            CALL SETBIT(.MWORD,4,1,1B);
 771    8                            FIELD(0) = 0FFH;
 772    8                            END;
 773    7                      END;
 774    6                  ELSE DO;
 775    7                      REGDECODE = ASCII2BIN(PHRASE(4));
 776    7                      IF FIELD(0) AND EQUBIT(.MWORD,4,1,1B)
                                  THEN DO;
 778    8                            E = E + 1;
 779    8                            CALL ALPHA(LP,37,.('*** CONFLICTING',
                                     ' SPECIFICATION FOR RS.'));
 780    8                            CALL EOL;
 781    8                            GOTO END1;
 782    8                            END;
 783    7                         ELSE DO;
 784    8                            CALL SETBIT(.MWORD,4,1,0B);
 785    8                            FIELD(0) = 0FFH;
 786    8                            END;
 787    7                      END;
 788    6               IF BADREG
                           THEN DO;
 790    7                      E = E + 1;
 791    7                      CALL ALPHA(LP,19,.('*** INVALID REG-ID.'));
 792    7                      CALL EOL;
 793    7                      GOTO END1;
 794    7                      END;
 795    6               IF FIELD(13) AND NOT EQUBIT(.MWORD,0,4,REGNO)
                           THEN DO;
 797    7                      E = E + 1;
 798    7                      CALL ALPHA(LP,23,.('*** CONFLICTING REG-ID.'));
 799    7                      CALL EOL;
 800    7                      GOTO END1;
 801    7                      END;
 802    6               IF EQUAL(.('ALC '),.PHRASE(7),4)        /* REG(R) = ALC */
                           THEN DO;
 804    7                      IF FIELD(16) AND NOT EQUBIT(.MWORD,9,1,1B)
```

```
                                    THEN DO;
806   8                                  E = E + 1;
807   8                                  CALL ALPHA(LP,53,.('*** CONFLICTING ',
                                         'SPECIFICATION FOR M2.  PHRASE IGNORED.'));
808   8                                  CALL EOL;
809   8                                  GOTO END1;
810   8                                  END;
811   7                               ELSE DO;
812   8                                  FIELD(13) = 0FFH;
813   8                                  CALL SETBIT(.MWORD,0,4,REGNO);
814   8                                  FIELD(16) = 0FFH;
815   8                                  CALL SETBIT(.MWORD,9,1,1B);
816   8                                  GOTO END1;
817   8                                  END;
818   7                            END;
819   6                      IF EQUAL(.('SHIFTER '),.PHRASE(7),8)
                                 THEN DO;
821   7                                  IF FIELD(13) AND NOT EQUBIT(.MWORD,9,1,0B)
                                         THEN DO;
823   8                                     E = E + 1;
824   8                                     CALL ALPHA(LP,53,.('*** CONFLICTING ',
                                            'SPECIFICATION FOR M2.  PHRASE IGNORED.'));
825   8                                     CALL EOL;
826   8                                     GOTO END1;
827   8                                     END;
828   7                                  ELSE DO;
829   8                                     FIELD(13), FIELD(16) = 0FFH;
830   8                                     CALL SETBIT(.MWORD,0,4,REGNO);
831   8                                     CALL SETBIT(.MWORD,9,1,0B);
832   8                                     GOTO END1;
833   8                                     END;
834   7                            END;
                        /*       IF NONE OF THE ABOVE THEN   */
835   6                      E = E + 1;
836   6                      CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
837   6                      CALL ALPHA(LP,32,.PHRASE);
838   6                      CALL EOL;
839   6                    END1:END;
```

```
                $EJECT

                /******************/
                /* 2: P1B-PHRASE */
                /******************/
 840   5                DO;
 841   6                DECLARE (REGNO,BADREG) BYTE;
 842   6                DECLARE REGDECODE ADDRESS AT(.REGNO);
 843   6                IF EQUAL(.('REG('),.PHRASE(4),4) AND
                             EQUAL(.(')'   ),.PHRASE(9),1)
                             THEN DO;
 845   7                     IF PHRASE(8) = '#'
                                THEN DO;
 847   8                          IF FIELD(0) AND EQUBIT(.MWORD,4,1,0B)
                                      THEN DO;
 849   9                               E = E + 1;
 850   9                               CALL ALPHA(LP,37,.('*** CON',
                                       'FLICTING SPECIFICATION FOR',
                                       ' RS.'));
 851   9                               CALL EOL;
 852   9                               GOTO END2;
 853   9                               END;
 854   8                          ELSE DO;
 855   9                               REGNO  = 0;
 856   9                               BADREG = 0;
 857   9                               CALL SETBIT(.MWORD,4,1,1B);
 858   9                               FIELD(0) = 0FFH;
 859   9                               END;
 860   8                          END;
 861   7                     ELSE DO;
 862   8                          REGDECODE = ASCII2BIN(PHRASE(8));
 863   8                          IF FIELD(0) AND EQUBIT(.MWORD,4,1,1B)
                                      THEN DO;
 865   9                               E = E + 1;
 866   9                               CALL ALPHA(LP,37,.('*** CON',
                                       'FLICTING SPECIFICATION FOR',
                                       ' RS.'));
 867   9                               CALL EOL;
 868   9                               GOTO END2;
 869   9                               END;
 870   8                          ELSE DO;
 871   9                               CALL SETBIT(.MWORD,4,1,0B);
 872   9                               FIELD(0) = 0FFH;
 873   9                               END;
 874   8                          END;
 875   7                     IF BADREG
                                THEN DO;
 877   8                          E = E + 1;
 878   8                          CALL ALPHA(LP,18,.('*** INVALID REG ID'));
 879   8                          CALL EOL;
 880   8                          GOTO END2;
 881   8                          END;
 882   7                     IF FIELD(13) AND NOT EQUBIT(.MWORD,0,4,REGNO)
                                THEN DO;
 884   8                          E = E + 1;
 885   8                          CALL ALPHA(LP,23,.('*** CONFLICT',
                                       'ING REG-ID.'));
```

```
886   8                              CALL EOL;
887   8                              GOTO END2;
888   8                              END;
889   7                         ELSE DO;
890   8                              FIELD(13) = 0FFH;
891   8                              CALL SETBIT(.MWORD,0,4,REGNO);
892   8                              END;
893   7                    IF FIELD(11) AND NOT EQUBIT(.MWORD,18,1,0B)
                              THEN DO;
895   8                              E = E + 1;
896   8                              CALL ALPHA(LP,53,.('*** CONFLICTING SPEC',
                                     'IFICATION FOR S1.  PHRASE IGNORED.'));
897   8                              CALL EOL;
898   8                              GOTO END2;
899   8                              END;
900   7                         ELSE DO;
901   8                              CALL SETBIT(.MWORD,18,2,10B);
902   8                              FIELD(11) = 0FFH;
903   8                              GOTO END2;
904   8                              END;
905   7                    END;
906   6               IF EQUAL(.('DI '),.PHRASE(4),3)
                        THEN DO;
908   7                    IF FIELD(11) AND NOT EQUBIT(.MWORD,18,1,0B)
                              THEN DO;
910   8                              E = E + 1;
911   8                              CALL ALPHA(LP,53,.('*** CONFLICTING ',
                                     'SPECIFICATION FOR S1.  PHRASE IGNORED.'));
912   8                              CALL EOL;
913   8                              GOTO END2;
914   8                              END;
915   7                         ELSE DO;
916   8                              FIELD(11) = 0FFH;
917   8                              P1BDIPHRASEPRESENT = 0FFH;
918   8                              CALL SETBIT(.MWORD,18,2,01B);
919   8                              GOTO END2;
920   8                              END;
921   7                    END;
922   6               IF EQUAL(.('@ '),.PHRASE(4),2)
                        THEN DO;
924   7                    IF FIELD(11) AND NOT EQUBIT(.MWORD,18,1,0B)
                              THEN DO;
926   8                              E = E + 1;
927   8                              CALL ALPHA(LP,53,.('*** CONFLICTING ',
                                     'SPECIFICATION FOR S1.  PHRASE IGNORED.'));
928   8                              CALL EOL;
929   8                              GOTO END2;
930   8                              END;
931   7                         ELSE DO;
932   8                              FIELD(11) = 0FFH;
933   8                              P1BDIPHRASEPRESENT = 0FFH;
934   8                              CALL SETBIT(.MWORD,18,2,01B);
935   8                              CALL SETBIT(.MWORD,20,1,1B);
936   8                              GOTO END2;
937   8                              END;
938   7                    END;
                    /*       IF NONE OF THE ABOVE THEN        */
```

```
939   6             E = E + 1;
940   6             CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
941   6             CALL ALPHA(LP,20,.PHRASE);
942   6             CALL EOL;
943   6           END2:END;
```

```
                $EJECT

                /********************/
                /* 3: P2B-PHRASE */
                /********************/
944    5                DO;
945    6                DECLARE (REGNO,BADREG) BYTE;
946    6                DECLARE REGDECODE ADDRESS AT(.REGNO);
947    6                IF EQUAL(.('P2B '),.PHRASE(4),4)
                           THEN DO;
949    7                       IF (FIELD(11) AND NOT EQUBIT(.MWORD,18,1,0B)) OR
                              (FIELD( 7) AND NOT EQUBIT(.MWORD,16,1,0B)) OR
                              (FIELD( 8) AND NOT EQUBIT(.MWORD,17,1,1B))
                              THEN DO;
951    8                             E = E + 1;
952    8                             CALL ALPHA(LP,58,.('*** CONFLICTING ',
                                     'SPECIFICATION FOR A OR S1.  PHRASE ',
                                     'IGNORED.  '));
953    8                             CALL EOL;
954    8                             GO TO END3;
955    8                             END;
956    7                       ELSE DO;
957    8                             FIELD(7), FIELD(8) = 0FFH;
958    8                             CALL SETBIT(.MWORD,16,2,01B);
959    8                             FIELD(11) = 0FFH;
960    8                             CALL SETBIT(.MWORD,18,2,01B);
961    8                             GOTO END3;
962    8                             END;
963    7                       END;
964    6                IF EQUAL(.('DI '),.PHRASE(4),3)
                           THEN DO;
966    7                       IF (FIELD(11) AND NOT EQUBIT(.MWORD,18,1,1B))  OR
                              (FIELD(7) AND FIELD(8)
                                          AND EQUBIT(.MWORD,16,2,01B))
                              THEN DO;
968    8                             E = E + 1;
969    8                             CALL ALPHA(LP,59,.('*** CONFLICTING ',
                                     'SPECIFICATION FOR A OR S1.  PHRASE ',
                                     'IGNORED.  '));
970    8                             CALL EOL;
971    8                             GOTO END3;
972    8                             END;
973    7                       ELSE DO;
974    8                             NOTAD1 = 0FFH;
975    8                             FIELD(11) = 0FFH;
976    8                             CALL SETBIT(.MWORD,18,2,10B);
977    8                             GOTO END3;
978    8                             END;
979    7                       END;.
980    6                IF EQUAL(.('@ '),.PHRASE(4),2)
                           THEN DO;
982    7                       IF (FIELD(11) AND NOT EQUBIT(.MWORD,18,1,1B)) OR
                              (FIELD( 7) AND FIELD(8)
                                          AND EQUBIT(.MWORD,16,2,01B))
                              THEN DO;
984    8                             E = E + 1;
985    8                             CALL ALPHA(LP,59,.('*** CONFLICTING ',
```

```
                                        'SPECIFICATION FOR A OR S1.  PHRASE ',
                                        'IGNORED.  '));
 986   8                                CALL EOL;
 987   8                                GOTO END3;
 988   8                                END;
 989   7                           ELSE DO;
 990   8                                NOTAD1 = 0FFH;
 991   8                                FIELD(11) = 0FFH;
 992   8                                CALL SETBIT(.MWORD,18,2,10B);
 993   8                                CALL SETBIT(.MWORD,20,1,1B);
 994   8                                GOTO END3;
 995   8                                END;
 996   7                      END;
 997   6                IF EQUAL(.('REG('),.PHRASE(4),4) AND
                           EQUAL(.('  ) '  '),.PHRASE(9),2)
                           THEN DO;
 999   7                      IF PHRASE(8) = '#'
                                THEN DO;
1001   8                           IF FIELD(0) AND EQUBIT(.MWORD,4,1,0B)
                                       THEN DO;
1003   9                                E = E + 1;
1004   9                                CALL ALPHA(LP,37,.('*** CON'),
                                        'FLICTING SPECIFICATION FOR'),
                                        ' RS. '));
1005   9                                CALL EOL;
1006   9                                GOTO END3;
1007   9                                END;
1008   8                           ELSE DO;
1009   9                                REGNO  = 0;
1010   9                                BADREG = 0;
1011   9                                CALL SETBIT(.MWORD,4,1,1B);
1012   9                                FIELD(0) = 0FFH;
1013   9                                END;
1014   8                           END;
1015   7                      ELSE DO;
1016   8                           REGDECODE = ASCII2BIN(PHRASE(8));
1017   8                           IF FIELD(0) AND EQUBIT(.MWORD,4,1,1B)
                                       THEN DO;
1019   9                                E = E + 1;
1020   9                                CALL ALPHA(LP,37,.('*** CON'),
                                        'FLICTING SPECIFICATION FOR'),
                                        ' RS. '));
1021   9                                CALL EOL;
1022   9                                GOTO END3;
1023   9                                END;
1024   8                           ELSE DO;
1025   9                                CALL SETBIT(.MWORD,4,1,0B);
1026   9                                FIELD(0) = 0FFH;
1027   9                                END;
1028   8                           END;
1029   7                      IF BADREG OR (REGNO > 7)
                                THEN DO;
1031   8                           E = E + 1;
1032   8                           CALL ALPHA(LP,18,.('*** INVALID REG-ID'));
1033   8                           CALL EOL;
1034   8                           GOTO END3;
1035   8                           END;
```

```
1036   7                          ELSE DO;
1037   8                               CALL SETBIT(.MWORD,5,3,REGNO);
1038   8                               FIELD(14) = 0FFH;
1039   8                               END;
1040   7                          P2BREGPHRASEPRESENT = 0FFH;
1041   7                          GOTO END3;
1042   7                          END;
                     /*       IF NONE OF THE ABOVE THEN                */
1043   6                       E = E + 1;
1044   6                       CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1045   6                       CALL ALPHA(LP,20,.PHRASE);
1046   6                       CALL EOL;
1047   6                     END3:END;
```

```
                $EJECT

                /*****************/
                /* 4: DO-PHRASE */
                /*****************/
1048    5               DO;
1049    6               IF EQUAL(.('OFF '),.PHRASE(3),4)
                           THEN DO;
1051    7                       CALL SETBIT(.MWORD,8,1,0B);
1052    7                       GOTO END4;
1053    7                       END;
1054    6               IF EQUAL(.('ALC '),.PHRASE(3),4)
                           THEN DO;
1056    7                       IF FIELD(16) AND EQUBIT(.MWORD, 9,1,1B) AND
                                   FIELD(17) AND EQUBIT(.MWORD,10,1,1B)
                                   THEN DO;
1058    8                              E = E + 1;
1059    8                              CALL ALPHA(LP,53,.('*** CONFLICTING '
                                       'SPECIFICATION FOR M. PHRASE IGNORED.'));
1060    8                              CALL EOL;
1061    8                              GOTO END4;
1062    8                              END;
1063    7                          ELSE DO;
1064    8                              DOALCPHRASEPRESENT = 0FFH;
1065    8                              CALL SETBIT(.MWORD,8,1,1B);
1066    8                              GOTO END4;
1067    8                              END;
1068    7                       END;
1069    6               IF EQUAL(.('P1B '),.PHRASE(3),4)
                           THEN DO;
1071    7                       IF FIELD(16) AND NOT EQUBIT(.MWORD, 9,1,1B) OR
                                   FIELD(17) AND NOT EQUBIT(.MWORD,10,1,1B) OR
                                   FIELD(18) AND NOT EQUBIT(.MWORD,11,1,1B)
                                   THEN DO;
1073    8                              E = E + 1;
1074    8                              CALL ALPHA(LP,53,.('*** CONFLICTING '
                                       'SPECIFICATION FOR M. PHRASE IGNORED.'));
1075    8                              CALL EOL;
1076    8                              GOTO END4;
1077    8                              END;
1078    7                          ELSE DO;
1079    8                              FIELD(16), FIELD(17), FIELD(18) = 0FFH;
1080    8                              CALL SETBIT(.MWORD,9,3,111B);
1081    8                              CALL SETBIT(.MWORD,8,1,1B);
1082    8                              GOTO END4;
1083    8                              END;
1084    7                       END;
1085    6               IF EQUAL(.('P2B '),.PHRASE(3),4)
                           THEN DO;
1087    7                       IF FIELD(16) AND NOT EQUBIT(.MWORD, 9,1,1B) OR
                                   FIELD(17) AND NOT EQUBIT(.MWORD,10,1,1B) OR
                                   FIELD(18) AND NOT EQUBIT(.MWORD,11,1,0B)
                                   THEN DO;
1089    8                              E = E + 1;
1090    8                              CALL ALPHA(LP,53,.('*** CONFLICTING '
                                       'SPECIFICATION FOR M. PHRASE IGNORED.'));
1091    8                              CALL EOL;
```

```
1092   8                              GOTO END4;
1093   8                              END;
1094   7                         ELSE DO;
1095   8                              FIELD(16), FIELD(17), FIELD(18) = 0FFH;
1096   8                              CALL SETBIT(.MWORD,9,3,110B);
1097   8                              CALL SETBIT(.MWORD,8,1,1B);
1098   8                              GOTO END4;
1099   8                              END;
1100   7                    END;
                    /*    IF NONE OF THE ABOVE THEN                    */
1101   6                    E = E + 1;
1102   6                    CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1103   6                    CALL ALPHA(LP,32,.PHRASE);
1104   6                    CALL EOL;
1105   6                  END4:END;
```

```
                $EJECT

                /*******************/
                /* 5: ALC-PHRASE */
                /*******************/
1106    5               DO;
1107    6               I = 32;                         /* POINT TO END OF PHRASE */
1108    6               DO WHILE PHRASE(I)=' ';         /* UNTIL LAST CHAR */
1109    7               I = I - 1;
1110    7               END;
                      /* NOW PHRASE(I) IS LAST NON-BLANK CHAR IN PHRASE */
1111    6               IF EQUAL(.('-P2B'),.PHRASE+I-3,4)
                           THEN DO;
1113    7                       IF I<25
                                  THEN DO;
1115    8                               PHRASE(I-3) = '+';
1116    8                               PHRASE(I-2) = '.';
1117    8                               PHRASE(I-1) = 'N';
1118    8                               PHRASE(I  ) = 'O';
1119    8                               PHRASE(I+1) = 'T';
1120    8                               PHRASE(I+2) = '.';
1121    8                               PHRASE(I+3) = 'P';
1122    8                               PHRASE(I+4) = '2';
1123    8                               PHRASE(I+5) = 'B';
1124    8                               PHRASE(I+6) = '+';
1125    8                               PHRASE(I+7) = 'C';
1126    8                               PHRASE(I+8) = 'O';
1127    8                               I = I + 8;
1128    8                               END;
1129    7                          ELSE DO;
1130    8                               E = E + 1;
1131    8                               CALL ALPHA(LP,20,.('*** ILLEGAL',
                                                           ' PHRASE: '));
1132    8                               CALL ALPHA(LP,32,.PHRASE);
1133    8                               CALL EOL;
1134    8                               GOTO END5;
1135    8                               END;
1136    7                       END;
1137    6               IF EQUAL(.('+CO'),.PHRASE+I-2,3)
                           THEN DO;
1139    7                       I = I - 3;
1140    7                       CALL SETBIT(.MWORD,26,2,01B);
1141    7                       END;
1142    6               IF EQUAL(.('+COUT'),.PHRASE+I-4,5)
                           THEN DO;
1144    7                       I = I - 5;
1145    7                       CALL SETBIT(.MWORD,26,2,10B);
1146    7                       END;
1147    6               IF EQUAL(.('+.NOT.COUT'),.PHRASE+I-9,10)
                           THEN DO;
1149    7                       I = I - 10;
1150    7                       CALL SETBIT(.MWORD,26,2,11B);
1151    7                       END;
1152    6               IF EQUAL(.('0'),.PHRASE+I,1)
                           THEN DO;
1154    7                       IF FIELD(20) AND NOT EQUBIT(.MWORD,13,1,1B)
                                   THEN DO;
```

```
1156   8                              E = E + 1;
1157   8                              CALL ALPHA(LP,52,.('*** CONFLICTING ',
                                      'SPECIFICATION FOR D.  PHRASE IGNORED.'));
1158   8                              CALL EOL;
1159   8                              GOTO END5;
1160   8                              END;
1161   7                         ELSE DO;
1162   8                              FIELD(20) = 0FFH;
1163   8                              CALL SETBIT(.MWORD,13,1,1B);
1164   8                              I = I - 1;
1165   8                              GOTO ENDRIGHT;
1166   8                              END;
1167   7                    END;
1168   6               IF EQUAL(.('.NOT.P2B'),.PHRASE+I-7,8)
                           THEN DO;
1170   7                         IF (FIELD(19) AND NOT EQUBIT(.MWORD,12,1,0B)) OR
                                    (FIELD(20) AND NOT EQUBIT(.MWORD,13,1,0B))
                                    THEN DO;
1172   8                              E = E + 1;
1173   8                              CALL ALPHA(LP,52,.('*** CONFLICTING ',
                                      'SPECIFICATION FOR D.  PHRASE IGNORED.'));
1174   8                              CALL EOL;
1175   8                              GOTO END5;
1176   8                              END;
1177   7                         ELSE DO;
1178   8                              FIELD(19), FIELD(20) = 0FFH;
1179   8                              CALL SETBIT(.MWORD,12,2,00B);
1180   8                              I = I - 8;
1181   8                              GOTO ENDRIGHT;
1182   8                              END;
1183   7                    END;
1184   6               IF EQUAL(.('P2B'),.PHRASE+I-2,3)
                           THEN DO;
1186   7                         IF (FIELD(19) AND NOT EQUBIT(.MWORD,12,1,1B)) OR
                                    (FIELD(20) AND NOT EQUBIT(.MWORD,13,1,0B))
                                    THEN DO;
1188   8                              E = E + 1;
1189   8                              CALL ALPHA(LP,52,.('*** CONFLICTING ',
                                      'SPECIFICATION FOR D.  PHRASE IGNORED.'));
1190   8                              CALL EOL;
1191   8                              GOTO END5;
1192   8                              END;
1193   7                         ELSE DO;
1194   8                              FIELD(19), FIELD(20) = 0FFH;
1195   8                              CALL SETBIT(.MWORD,12,2,10B);
1196   8                              I = I - 3;
1197   8                              GOTO ENDRIGHT;
1198   8                              END;
1199   7                    END;
                      /*       IF NONE OF THE ABOVE THEN                */
1200   6                    E = E + 1;
1201   6                    CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1202   6                    CALL ALPHA(LP,32,.PHRASE);
1203   6                    CALL EOL;
1204   6                    GOTO END5;
1205   6               ENDRIGHT:
                           IF EQUAL(.('P1B'),.PHRASE(4),3) AND I>6
```

```
                        THEN DO;
1207   7                     IF NOT FIELD(19) OR  EQUBIT(.MWORD,12,2,10B)
                                THEN DO;
1209   8                           FIELD(19) = 0FFH;
1210   8                           CALL SETBIT(.MWORD,12,1,1B);
1211   8                           GOTO ENDLEFT;
1212   8                           END;
1213   7                     IF FIELD(19) AND EQUBIT(.MWORD,12,2,00B)
                                THEN DO;
                                    /* OK AS IS */
1215   8                           GOTO ENDLEFT;
1216   8                           END;
1217   7                     END;
1218   6                IF EQUAL(.('.NOT.P1B'),.PHRASE(4),8) AND I>11
                            THEN DO;
1220   7                     IF (FIELD(19) AND EQUBIT(.MWORD,12,1,1B)) OR
                                (FIELD(20) AND EQUBIT(.MWORD,13,1,0B))
                              THEN DO;
1222   8                           E = E + 1;
1223   8                           CALL ALPHA(LP,52,.('*** CONFLICT',
                                    'ING SPECIFICATION FOR D.  PHRASE',
                                    ' IGNORED.'));
1224   8                           CALL EOL;
1225   8                           GOTO END5;
1226   8                           END;
1227   7                     ELSE DO;
1228   8                           FIELD(19), FIELD(20) = 0FFH;
1229   8                           CALL SETBIT(.MWORD,12,2,01B);
1230   8                           GOTO ENDLEFT;
1231   8                           END;
1232   7                     END;
                     /*       IF NONE OF THE ABOVE THEN                  */
1233   6                E = E + 1;
1234   6                CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1235   6                CALL ALPHA(LP,32,.PHRASE);
1236   6                CALL EOL;
1237   6                GOTO END5;
1238   6           ENDLEFT:
                         IF EQUAL(.('+'),.PHRASE+I,1)
                            THEN DO;
1240   7                     IF FIELD(7) AND NOT EQUBIT(.MWORD,16,1,0B)
                                THEN DO;
1242   8                           E = E + 1;
1243   8                           CALL ALPHA(LP,52,.('*** CONFLICTING ',
                                    'SPECIFICATION FOR A.  PHRASE IGNORED.'));
1244   8                           CALL EOL;
1245   8                           GOTO END5;
1246   8                           END;
1247   7                     ELSE DO;
1248   8                           FIELD(7) = 0FFH;
1249   8                           CALL SETBIT(.MWORD,16,1,0B);
1250   8                           GOTO END5;
1251   8                           END;
1252   7                     END;
1253   6                IF EQUAL(.('OR'),.PHRASE+I-1,2)
                            THEN DO;
1255   7                     IF FIELD(7) AND NOT EQUBIT(.MWORD,16,1,1B) OR
```

```
                                FIELD(8) AND NOT EQUBIT(.MWORD,17,1,1B)
                                THEN DO;
1257   8                             E = E + 1;
1258   8                             CALL ALPHA(LP,52,.('*** CONFLICTING ',
                                     'SPECIFICATION FOR A.  PHRASE IGNORED.  '));
1259   8                             CALL EOL;
1260   8                             GOTO END5;
1261   8                             END;
1262   7                          ELSE DO;
1263   8                             FIELD(7), FIELD(8) = 0FFH;
1264   8                             CALL SETBIT(.MWORD,16,2,11B);
1265   8                             GOTO END5;
1266   8                             END;
1267   7                       END;
1268   6                IF EQUAL(.('AND'),.PHRASE+I-2,3)
                          THEN DO;
1270   7                       IF FIELD(7) AND NOT EQUBIT(.MWORD,16,1,1B) OR
                                FIELD(8) AND NOT EQUBIT(.MWORD,17,1,0B)
                                THEN DO;
1272   8                             E = E + 1;
1273   8                             CALL ALPHA(LP,52,.('*** CONFLICTING ',
                                     'SPECIFICATION FOR A.  PHRASE IGNORED.  '));
1274   8                             CALL EOL;
1275   8                             GOTO END5;
1276   8                             END;
1277   7                          ELSE DO;
1278   8                             FIELD(7), FIELD(8) = 0FFH;
1279   8                             CALL SETBIT(.MWORD,16,2,10B);
1280   8                             GOTO END5;
1281   8                             END;
1282   7                       END;
                     /*       IF NONE OF THE ABOVE THEN                */
1283   6                E = E + 1;
1284   6                CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1285   6                CALL ALPHA(LP,32,.PHRASE);
1286   6                CALL EOL;
1287   6              END5:END;
```

```
                $EJECT

                /**********************/
                /* 6: SHIFTER-PHRASE */
                /**********************/
1288    5               DO;
1289    6               IF EQUAL(.('@ '),.PHRASE(8),2)
                           THEN DO;
1291    7                       IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,0B) OR
                                   FIELD(18) AND NOT EQUBIT(.MWORD,11,1,0B) OR
                                   FIELD(11) AND NOT EQUBIT(.MWORD,18,1,0B)
                                   THEN DO;
1293    8                              E = E + 1;
1294    8                              CALL ALPHA(LP,57,.('*** CONFLICTING ',
                                       'SPECIFICATION FOR M OR S.  PHRASE IG',
                                       'NORED. '));
1295    8                              CALL EOL;
1296    8                              GOTO END6;
1297    8                              END;
1298    7                          ELSE DO;
1299    8                              FIELD(17), FIELD(18), FIELD(11) = 0FFH;
1300    8                              CALL SETBIT(.MWORD,10,2,00B);
1301    8                              CALL SETBIT(.MWORD,18,2,01B);
1302    8                              CALL SETBIT(.MWORD,20,1,1B);
1303    8                              GOTO END6;
1304    8                              END;
1305    7                       END;
1306    6               IF EQUAL(.('DI '),.PHRASE(8),3)
                           THEN DO;
1308    7                       IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,0B) OR
                                   FIELD(18) AND NOT EQUBIT(.MWORD,11,1,0B) OR
                                   FIELD(11) AND NOT EQUBIT(.MWORD,18,1,0B)
                                   THEN DO;
1310    8                              E = E + 1;
1311    8                              CALL ALPHA(LP,57,.('*** CONFLICTING ',
                                       'SPECIFICATION FOR M OR S.  PHRASE IG',
                                       'NORED. '));
1312    8                              CALL EOL;
1313    8                              GOTO END6;
1314    8                              END;
1315    7                          ELSE DO;
1316    8                              FIELD(17), FIELD(18), FIELD(11) = 0FFH;
1317    8                              CALL SETBIT(.MWORD,10,2,00B);
1318    8                              CALL SETBIT(.MWORD,18,2,01B);
1319    8                              GOTO END6;
1320    8                              END;
1321    7                       END;
1322    6               IF EQUAL(.('ALC(6-0)\\0 '),.PHRASE(8),12)
                           THEN DO;
1324    7                       IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,0B) OR
                                   FIELD(18) AND NOT EQUBIT(.MWORD,11,1,1B) OR
                                   FIELD( 2) AND NOT EQUBIT(.MWORD,14,1,1B) OR
                                   FIELD( 3) AND NOT EQUBIT(.MWORD,15,1,1B)
                                   THEN DO;
1326    8                              E = E + 1;
1327    8                              CALL ALPHA(LP,57,.('*** CONFLICTING ',
                                       'SPECIFICATION FOR M OR C.  PHRASE IG',
```

```
                                        'NORED.'));
1328   8                                CALL EOL;
1329   8                                GOTO END6;
1330   8                                END;
1331   7                           ELSE DO;
1332   8                                FIELD( 2), FIELD( 3),
                                                FIELD(17), FIELD(18) = 0FFH;
1333   8                                CALL SETBIT(.MWORD,10,2,01B);
1334   8                                CALL SETBIT(.MWORD,14,2,11B);
1335   8                                GOTO END6;
1336   8                                END;
1337   7                      END;
1338   6                IF EQUAL(.('ALC(6-0)\\1 '),.PHRASE(8),12)
                           THEN DO;
1340   7                      IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,0B) OR
                              FIELD(18) AND NOT EQUBIT(.MWORD,11,1,1B) OR
                              FIELD( 2) AND NOT EQUBIT(.MWORD,14,1,1B) OR
                              FIELD( 3) AND NOT EQUBIT(.MWORD,15,1,0B)
                              THEN DO;
1342   8                           E = E + 1;
1343   8                           CALL ALPHA(LP,57,.('*** CONFLICTING '
                                   'SPECIFICATION FOR M OR C.  PHRASE IG'
                                   'NORED.'));
1344   8                           CALL EOL;
1345   8                           GOTO END6;
1346   8                           END;
1347   7                      ELSE DO;
1348   8                           FIELD( 2), FIELD( 3),
                                                FIELD(17), FIELD(18) = 0FFH;
1349   8                           CALL SETBIT(.MWORD,10,2,01B);
1350   8                           CALL SETBIT(.MWORD,14,2,10B);
1351   8                           GOTO END6;
1352   8                           END;
1353   7                      END;
1354   6                IF EQUAL(.('ALC(6-0)\\MXL0IN '),.PHRASE(8),17)
                           THEN DO;
1356   7                      IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,0B) OR
                              FIELD(18) AND NOT EQUBIT(.MWORD,11,1,1B)
                              THEN DO;
1358   8                           E = E + 1;
1359   8                           CALL ALPHA(LP,52,.('*** CONFLICTING '
                                   'SPECIFICATION FOR M.  PHRASE IGNORED.'));
1360   8                           CALL EOL;
1361   8                           GOTO END6;
1362   8                           END;
1363   7                      ELSE DO;
1364   8                           FIELD(17), FIELD(18) = 0FFH;
1365   8                           CALL SETBIT(.MWORD,10,2,01B);
1366   8                           GOTO END6;
1367   8                           END;
1368   7                      END;
1369   6                IF EQUAL(.('0\\ALC(7-1) '),.PHRASE(8),12)
                           THEN DO;
1371   7                      IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,1B) OR
                              FIELD(18) AND NOT EQUBIT(.MWORD,11,1,0B) OR
                              FIELD( 2) AND NOT EQUBIT(.MWORD,14,1,1B) OR
                              FIELD( 3) AND NOT EQUBIT(.MWORD,15,1,1B)
```

```
                                    THEN DO;
1373   8                                 E = E + 1;
1374   8                                 CALL ALPHA(LP,57,.('*** CONFLICTING ',
                                         'SPECIFICATION FOR M OR C.  PHRASE IG',
                                         'NORED. '));
1375   8                                 CALL EOL;
1376   8                                 GOTO END6;
1377   8                                 END;
1378   7                              ELSE DO;
1379   8                                 FIELD( 2), FIELD( 3),
                                                 FIELD(17), FIELD(18) = 0FFH;
1380   8                                 CALL SETBIT(.MWORD,10,2,10B);
1381   8                                 CALL SETBIT(.MWORD,14,2,11B);
1382   8                                 GOTO END6;
1383   8                                 END;
1384   7                           END;
1385   6                     IF EQUAL(.('1\\ALC(7-1) '),.PHRASE(8),12)
                              THEN DO;
1387   7                           IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,1B) OR
                                    FIELD(18) AND NOT EQUBIT(.MWORD,11,1,0B) OR
                                    FIELD( 2) AND NOT EQUBIT(.MWORD,14,1,1B) OR
                                    FIELD( 3) AND NOT EQUBIT(.MWORD,15,1,0B)
                                    THEN DO;
1389   8                                 E = E + 1;
1390   8                                 CALL ALPHA(LP,57,.('*** CONFLICTING ',
                                         'SPECIFICATION FOR M OR C.  PHRASE IG',
                                         'NORED. '));
1391   8                                 CALL EOL;
1392   8                                 GOTO END6;
1393   8                                 END;
1394   7                              ELSE DO;
1395   8                                 FIELD( 2), FIELD( 3),
                                                 FIELD(17), FIELD(18) = 0FFH;
1396   8                                 CALL SETBIT(.MWORD,10,2,10B);
1397   8                                 CALL SETBIT(.MWORD,14,2,10B);
1398   8                                 GOTO END6;
1399   8                                 END;
1400   7                           END;
1401   6                     IF EQUAL(.('MXH0IN\\ALC(7-1) '),.PHRASE(8),17)
                              THEN DO;
1403   7                           IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,1B) OR
                                    FIELD(18) AND NOT EQUBIT(.MWORD,11,1,0B)
                                    THEN DO;
1405   8                                 E = E + 1;
1406   8                                 CALL ALPHA(LP,52,.('*** CONFLICTING ',
                                         'SPECIFICATION FOR M.  PHRASE IGNORED.'));
1407   8                                 CALL EOL;
1408   8                                 GOTO END6;
1409   8                                 END;
1410   7                              ELSE DO;
1411   8                                 FIELD(17), FIELD(18) = 0FFH;
1412   8                                 CALL SETBIT(.MWORD,10,2,10B);
1413   8                                 GOTO END6;
1414   8                                 END;
1415   7                           END;
1416   6                     IF EQUAL(.('00\\ALC(7-2) '),.PHRASE(8),13)
                              THEN DO;
```

```
1418   7                    IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,1B) OR
                               FIELD(18) AND NOT EQUBIT(.MWORD,11,1,1B) OR
                               FIELD( 2) AND NOT EQUBIT(.MWORD,14,1,1B) OR
                               FIELD( 3) AND NOT EQUBIT(.MWORD,15,1,1B)
                               THEN DO;
1420   8                         E = E + 1;
1421   8                         CALL ALPHA(LP,57,.('*** CONFLICTING ',
                                 'SPECIFICATION FOR M OR C.  PHRASE IG',
                                 'NORED. '));
1422   8                         CALL EOL;
1423   8                         GOTO END6;
1424   8                         END;
1425   7                      ELSE DO;
1426   8                         FIELD( 2), FIELD( 3),
                                           FIELD(17), FIELD(18) = 0FFH;
1427   8                         CALL SETBIT(.MWORD,10,2,11B);
1428   8                         CALL SETBIT(.MWORD,14,2,11B);
1429   8                         GOTO END6;
1430   8                         END;
1431   7                   END;
1432   6             IF EQUAL(.('11\\ALC(7-2) '),.PHRASE(8),13)
                         THEN DO;
1434   7                      IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,1B) OR
                               FIELD(18) AND NOT EQUBIT(.MWORD,11,1,1B) OR
                               FIELD( 2) AND NOT EQUBIT(.MWORD,14,1,1B) OR
                               FIELD( 3) AND NOT EQUBIT(.MWORD,15,1,0B)
                               THEN DO;
1436   8                         E = E + 1;
1437   8                         CALL ALPHA(LP,57,.('*** CONFLICTING ',
                                 'SPECIFICATION FOR M OR C.  PHRASE IG',
                                 'NORED. '));
1438   8                         CALL EOL;
1439   8                         GOTO END6;
1440   8                         END;
1441   7                      ELSE DO;
1442   8                         FIELD( 2), FIELD( 3),
                                           FIELD(17), FIELD(18) = 0FFH;
1443   8                         CALL SETBIT(.MWORD,10,2,11B);
1444   8                         CALL SETBIT(.MWORD,14,2,10B);
1445   8                         GOTO END6;
1446   8                         END;
1447   7                   END;
1448   6             IF EQUAL(.('MXH1IN\\MXH0IN\\ALC(7-2)'),.PHRASE(8),24)
                         THEN DO;
1450   7                      IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,1B) OR
                               FIELD(18) AND NOT EQUBIT(.MWORD,11,1,1B)
                               THEN DO;
1452   8                         E = E + 1;
1453   8                         CALL ALPHA(LP,52,.('*** CONFLICTING ',
                                 'SPECIFICATION FOR M.  PHRASE IGNORED. '));
1454   8                         CALL EOL;
1455   8                         GOTO END6;
1456   8                         END;
1457   7                      ELSE DO;
1458   8                         FIELD(17), FIELD(18) = 0FFH;
1459   8                         CALL SETBIT(.MWORD,10,2,11B);
1460   8                         GOTO END6;
```

```
1461   8                              END;
1462   7                        END;
                  /*      IF NONE OF THE ABOVE THEN              */
1463   6                     E = E + 1;
1464   6                     CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1465   6                     CALL ALPHA(LP,32,.PHRASE);
1466   6                     CALL EOL;
1467   6                  END6:END;
```

```
                $EJECT

                /************************/
                /*  7: MXL0OUT-PHRASE */
                /************************/
1468    5               DO;
1469    6               IF EQUAL(.('ALC(0) '),.PHRASE(8),7)
                           THEN DO;
1471    7                       IF FIELD(16) AND NOT EQUBIT(.MWORD, 9,1,0B) OR
                                   FIELD(17) AND NOT EQUBIT(.MWORD,10,1,1B)
                                   THEN DO;
1473    8                                  E = E + 1;
1474    8                                  CALL ALPHA(LP,53,.('*** CONFLICTING '
                                           'SPECIFICATION FOR M.  PHRASE IGNORED.')));
1475    8                                  CALL EOL;
1476    8                                  GOTO END7;
1477    8                                  END;
1478    7                              ELSE DO;
1479    8                                  FIELD(16), FIELD(17) = 0FFH;
1480    8                                  CALL SETBIT(.MWORD,9,2,01B);
1481    8                                  GOTO END7;
1482    8                                  END;
1483    7                           END;
1484    6                       ELSE DO;
1485    7                           E = E + 1;
1486    7                           CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1487    7                           CALL ALPHA(LP,32,.PHRASE);
1488    7                           CALL EOL;
1489    7                           GOTO END7;
1490    7                           END;
1491    6           END7:END;
```

```
                $EJECT

                /************************/
                /*  8: MXL10UT-PHRASE */
                /************************/
1492    5               DO;
1493    6               IF EQUAL(.('ALC(1) '),.PHRASE(8),7)
                           THEN DO;
1495    7                       IF FIELD(16) AND NOT EQUBIT(.MWORD,9,1,0B) OR
                                   FIELD(17) AND NOT EQUBIT(.MWORD,10,1,1B) OR
                                   FIELD(18) AND NOT EQUBIT(.MWORD,11,1,1B)
                                   THEN DO;
1497    8                               E = E + 1;
1498    8                               CALL ALPHA(LP,53,.('*** CONFLICTING '
                                        'SPECIFICATION FOR M.  PHRASE IGNORED.'));
1499    8                               CALL EOL;
1500    8                               GOTO END8;
1501    8                               END;
1502    7                           ELSE DO;
1503    8                               FIELD(16),FIELD(17),FIELD(18) = 0FFH;
1504    8                               CALL SETBIT(.MWORD,9,3,011B);
1505    8                               GOTO END8;
1506    8                               END;
1507    7                       END;
1508    6                   ELSE DO;
1509    7                       E = E + 1;
1510    7                       CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1511    7                       CALL ALPHA(LP,32,.PHRASE);
1512    7                       CALL EOL;
1513    7                       END;
1514    6           END8:END;
```

```
                $EJECT

                /************************/
                /*  9: MXH0OUT-PHRASE */
                /************************/
1515    5               DO;
1516    6               IF EQUAL(.('ALC(7) '),.PHRASE(8),7)
                           THEN DO;
1518    7                       IF FIELD(16) AND NOT EQUBIT(.MWORD, 9,1,0B) OR
                                   FIELD(17) AND NOT EQUBIT(.MWORD,10,1,0B) OR
                                   FIELD(18) AND NOT EQUBIT(.MWORD,11,1,1B)
                                   THEN DO;
1520    8                              E = E + 1;
1521    8                              CALL ALPHA(LP,52,.('*** CONFLICTING ',
                                       'SPECIFICATION FOR M.  PHRASE IGNORED.'));
1522    8                              CALL EOL;
1523    8                              GOTO END9;
1524    8                              END;
1525    7                           ELSE DO;
1526    8                              FIELD(16), FIELD(17), FIELD(18) = 0FFH;
1527    8                              CALL SETBIT(.MWORD,9,3,001B);
1528    8                              GOTO END9;
1529    8                              END;
1530    7                       END;
1531    6                   ELSE DO;
1532    7                       E = E + 1;
1533    7                       CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1534    7                       CALL ALPHA(LP,32,.PHRASE);
1535    7                       CALL EOL;
1536    7                       END;
1537    6       END9:   END;
```

```
                $EJECT

                /************************/
                /* 10: MXH10UT-PHRASE */
                /************************/
1538    5               DO;
1539    6               IF EQUAL(.('OFL '),.PHRASE(8),4)
                           THEN DO;
1541    7                       IF FIELD(2) AND NOT EQUBIT(.MWORD,14,1,1B)
                                   THEN DO;
1543    8                               E = E + 1;
1544    8                               CALL ALPHA(LP,53,.('*** CONFLICTING',
                                        'SPECIFICATION FOR C.  PHRASE IGNORED.'));
1545    8                               CALL EOL;
1546    8                               END;
1547    7                           ELSE DO;
1548    8                               FIELD(2) = 0FFH;
1549    8                               CALL SETBIT(.MWORD,14,1,1B);
1550    8                               END;
1551    7                       END;
1552    6                   ELSE DO;
1553    7                       E = E + 1;
1554    7                       CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1555    7                       CALL ALPHA(LP,32,.PHRASE);
1556    7                       CALL EOL;
1557    7                       END;
1558    6           END10:END;
```

```
                $EJECT
                /*******************/
                /* 11: CO-PHRASES */
                /*******************/
1559    5               DO;
1560    6               IF EQUAL(.('GPUCOUT '),.PHRASE(3),8)
                           THEN DO;
1562    7                       CALL SETBIT(.MWORD,21,1,0B);
1563    7                       GOTO END11;
1564    7                       END;
1565    6               IF EQUAL(.('L2 '),.PHRASE(3),3)
                           THEN DO;
1567    7                       CALL SETBIT(.MWORD,21,1,1B);
1568    7                       GOTO END11;
1569    7                       END;
                      /* IF NONE OF THE ABOVE THEN */
1570    6               E = E + 1;
1571    6               CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1572    6               CALL ALPHA(LP,32,.PHRASE);
1573    6               CALL EOL;
1574    6         END11:END;
```

```
                $EJECT
                /************************/
                /* 12: MFLAGS-PHRASES */
                /************************/
1575   5                DO;
1576   6                IF EQUAL(.('DISABLE '),.PHRASE(7),8)
                           THEN DO;
1578   7                        CALL SETBIT(.MWORD,22,2,00B);
1579   7                        GOTO END12;
1580   7                        END;
1581   6                IF EQUAL(.('ENABLE(OUT) '),.PHRASE(7),11)
                           THEN DO;
1583   7                        CALL SETBIT(.MWORD,22,2,01B);
1584   7                        GOTO END12;
1585   7                        END;
1586   6                IF EQUAL(.('ENABLE(SZP) '),.PHRASE(7),11)
                           THEN DO;
1588   7                        CALL SETBIT(.MWORD,22,2,10B);
1589   7                        GOTO END12;
1590   7                        END;
1591   6                IF EQUAL(.('ENABLE(ALL) '),.PHRASE(7),11)
                           THEN DO;
1593   7                        CALL SETBIT(.MWORD,22,2,11B);
1594   7                        GOTO END12;
1595   7                        END;
                       /* IF NONE OF THE ABOVE THEN */
1596   6                   E = E + 1;
1597   6                   CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1598   6                   CALL ALPHA(LP,32,.PHRASE);
1599   6                   CALL EOL;
1600   6              END12:END;
```

```
                $EJECT
                /*********************/
                /* 13: COUT-PHRASES */
                /*********************/
1601    5               DO;
1602    6               IF EQUAL(.('COUT '),.PHRASE(5),5)
                           THEN DO;
1604    7                       CALL SETBIT(.MWORD,24,2,00B);
1605    7                       GOTO END13;
1606    7                       END;
1607    6               IF EQUAL(.('GPUCOUT '),.PHRASE(5),8)
                           THEN DO;
1609    7                       CALL SETBIT(.MWORD,24,2,01B);
1610    7                       GOTO END13;
1611    7                       END;
1612    6               IF EQUAL(.('DO(7) '),.PHRASE(5),6)
                           THEN DO;
1614    7                       CALL SETBIT(.MWORD,24,2,10B);
1615    7                       GOTO END13;
1616    7                       END;
1617    6               IF EQUAL(.('DO(0) '),.PHRASE(5),6)
                           THEN DO;
1619    7                       CALL SETBIT(.MWORD,24,2,11B);
1620    7                       GOTO END13;
1621    7                       END;
                      /* IF NONE OF THE ABOVE THEN */
1622    6                  E = E + 1;
1623    6                  CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1624    6                  CALL ALPHA(LP,32,.PHRASE);
1625    6                  CALL EOL;
1626    6           END13:END;
```

```
                $EJECT
                /**********************/
                /* 14: NOLOAD-PHRASE */
                /**********************/
1627   5                DO;
1628   6                IF FIELD(16) AND NOT EQUBIT(.MWORD, 9,1,1B) OR
                           FIELD(17) AND NOT EQUBIT(.MWORD,10,1,0B) OR
                           FIELD(18) AND NOT EQUBIT(.MWORD,11,1,0B)
                           THEN DO;
1630   7                        E = E + 1;
1631   7                        CALL ALPHA(LP,52,.('*** CONFLICTING SPECIFI',
                                'CATION FOR M. PHRASE IGNORED.'));
1632   7                        CALL EOL;
1633   7                        END;
1634   6                    ELSE DO;
1635   7                        CALL SETBIT(.MWORD,9,3,100B);
1636   7                        FIELD(16), FIELD(17), FIELD(18) = 0FFH;
1637   7                        END;
1638   6          END14:END;
```

```
                $EJECT
                /*********************/
                /* 15:  I/O-PHRASES */
                /*********************/
1639    5               DO;
1640    6               IF EQUAL(.('.NOT.MEMR '),.PHRASE(4),10)
                           THEN DO;
1642    7                       CALL SETBIT(.MWORD,28,4,0001B);
1643    7                       GOTO END15;
1644    7                       END;
1645    6               IF EQUAL(.('.NOT.MEMW '),.PHRASE(4),10)
                           THEN DO;
1647    7                       CALL SETBIT(.MWORD,28,4,0010B);
1648    7                       GOTO END15;
1649    7                       END;
1650    6               IF EQUAL(.('.NOT.IOR '),.PHRASE(4),9)
                           THEN DO;
1652    7                       CALL SETBIT(.MWORD,28,4,0011B);
1653    7                       GOTO END15;
1654    7                       END;
1655    6               IF EQUAL(.('.NOT.IOW '),.PHRASE(4),9)
                           THEN DO;
1657    7                       CALL SETBIT(.MWORD,28,4,0100B);
1658    7                       GOTO END15;
1659    7                       END;
1660    6               IF EQUAL(.('HLDA '),.PHRASE(4),5)
                           THEN DO;
1662    7                       CALL SETBIT(.MWORD,28,4,0101B);
1663    7                       GOTO END15;
1664    7                       END;
1665    6               IF EQUAL(.('WAIT '),.PHRASE(4),5)
                           THEN DO;
1667    7                       CALL SETBIT(.MWORD,28,4,0110B);
1668    7                       GOTO END15;
1669    7                       END;
1670    6               IF EQUAL(.('INTES '),.PHRASE(4),6)
                           THEN DO;
1672    7                       CALL SETBIT(.MWORD,28,4,0111B);
1673    7                       GOTO END15;
1674    7                       END;
1675    6               IF EQUAL(.('.NOT.INTA.NOT.IFCH '),.PHRASE(4),19)
                           THEN DO;
1677    7                       CALL SETBIT(.MWORD,28,4,1000B);
1678    7                       GOTO END15;
1679    7                       END;
1680    6               IF EQUAL(.('HLDA.WAIT '),.PHRASE(4),10)
                           THEN DO;
1682    7                       CALL SETBIT(.MWORD,28,4,1001B);
1683    7                       GOTO END15;
1684    7                       END;
1685    6               IF EQUAL(.('INTER '),.PHRASE(4),6)
                           THEN DO;
1687    7                       CALL SETBIT(.MWORD,28,4,1010B);
1688    7                       GOTO END15;
1689    7                       END;
1690    6               IF EQUAL(.('.NOT.MEMR.NOT.IFCH '),.PHRASE(4),19)
                           THEN DO;
```

```
1692   7                    CALL SETBIT(.MWORD,28,4,1011B);
1693   7                    GOTO END15;
1694   7                    END;
1695   6              IF EQUAL(.('.NOT.MEMR.NOT STACK '),.PHRASE(4),20)
                         THEN DO;
1697   7                    CALL SETBIT(.MWORD,28,4,1100B);
1698   7                    GOTO END15;
1699   7                    END;
1700   6              IF EQUAL(.('.NOT.MEMW.NOT.STACK '),.PHRASE(4),20)
                         THEN DO;
1702   7                    CALL SETBIT(.MWORD,28,4,1101B);
1703   7                    GOTO END15;
1704   7                    END;
1705   6              IF EQUAL(.('EI '),.PHRASE(4),3)
                         THEN DO;
1707   7                    CALL SETBIT(.MWORD,28,4,1110B);
1708   7                    GOTO END15;
1709   7                    END;
1710   6              IF EQUAL(.('DI '),.PHRASE(4),3)
                         THEN DO;
1712   7                    CALL SETBIT(.MWORD,28,4,1111B);
1713   7                    GOTO END15;
1714   7                    END;
                     /* IF NONE OF THE ABOVE THEN */
1715   6                 E = E + 1;
1716   6                 CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1717   6                 CALL ALPHA(LP,32,.PHRASE);
1718   6                 CALL EOL;
1719   6           END15:END;
```

```
                $EJECT
                /*******************/
                /* 16: GUA-PHRASES */
                /*******************/
1720   5                DO;
1721   6                IF EQUAL(.('ENABLE(CE) '),.PHRASE(4),11)
                           THEN DO;
1723   7                        CALL SETBIT(.MWORD,32,2,00B);
1724   7                        GOTO END16;
1725   7                        END;
1726   6                IF EQUAL(.('ENABLE(LE1) '),.PHRASE(4),12)
                           THEN DO;
1728   7                        CALL SETBIT(.MWORD,32,2,01B);
1729   7                        GOTO END16;
1730   7                        END;
1731   6                IF EQUAL(.('ENABLE(LE2) '),.PHRASE(4),12)
                           THEN DO;
1733   7                        CALL SETBIT( MWORD,32,2,10B);
1734   7                        GOTO END16;
1735   7                        END;
1736   6                IF EQUAL(.('ENABLE(LE3) '),.PHRASE(4),12)
                           THEN DO;
1738   7                        CALL SETBIT(.MWORD,32,2,11B);
1739   7                        GOTO END16;
1740   7                        END;
                      /* IF NONE OF THE ABOVE THEN */
1741   6                   E = E + 1;
1742   6                   CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1743   6                   CALL ALPHA(LP,32,.PHRASE);
1744   6                   CALL EOL;
1745   6             END16:END;
```

```
                $EJECT
                /*****************/
                /* 17: E-PHRASES */
                /*****************/
1746    5               DO;
1747    6               IF EQUAL(.('0 '),.PHRASE(2),2)
                           THEN DO;
1749    7                       CALL SETBIT(.MWORD,34,1,0B);
1750    7                       GOTO END17;
1751    7                       END;
1752    6               IF EQUAL(.('1 '),.PHRASE(2),2)
                           THEN DO;
1754    7                       CALL SETBIT(.MWORD,34,1,1B);
1755    7                       GOTO END17;
1756    7                       END;
                     /* IF NONE OF THE ABOVE THEN */
1757    6               E = E + 1;
1758    6               CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1759    6               CALL ALPHA(LP,32,.PHRASE);
1760    6               CALL EOL;
1761    6          END17:END;
```

```
                $EJECT
                /************************/
                /* 18: PHRASE(0) = ' ' */
                /************************/
1762    5
```

```
            $EJECT

            /************************/
            /* 19: PHRASE(0) = ';' */
            /************************/
1763   5
```

```
                $EJECT

                /**************/
                /* 20: OTHER */
                /**************/

1764   5                DO;
1765   6                IF NOT EQUAL(.('A='),.PHRASE,2) AND
                           NOT EQUAL(.('C='),.PHRASE,2) AND
                           NOT EQUAL(.('D='),.PHRASE,2) AND
                           NOT EQUAL(.('M='),.PHRASE,2) AND
                           NOT EQUAL(.('S='),.PHRASE,2) OR
                           PHRASE(2) < 30H OR PHRASE(2) > 37H OR
                           NOT EQUAL(.(' '),.PHRASE(3),1)
                           THEN DO;
1767   7                        E = E + 1;
1768   7                        CALL ALPHA(LP,20,.('*** ILLEGAL PHRASE: '));
1769   7                        CALL ALPHA(LP,32,.PHRASE);
1770   7                        CALL EOL;
1771   7                        END;
1772   6                   ELSE DO;
1773   7                        DECLARE Z BYTE;
1774   7                        Z = PHRASE(2) AND 07H;
1775   7                        IF EQUAL(.('A'),.PHRASE,1)
                                   THEN DO;
1777   8                             FIELD(7), FIELD(8) = 0FFH;
1778   8                             CALL SETBIT(.MWORD,16,2,Z);
1779   8                             END;
1780   7                        IF EQUAL(.('C'),.PHRASE,1)
                                   THEN DO;
1782   8                             FIELD(2), FIELD(3) = 0FFH;
1783   8                             CALL SETBIT(.MWORD,14,2,Z);
1784   8                             END;
1785   7                        IF EQUAL(.('D'),.PHRASE,1)
                                   THEN DO;
1787   8                             FIELD(19), FIELD(20) = 0FFH;
1788   8                             Z = SHR(Z,1);
1789   8                             CALL SETBIT(.MWORD,12,2,Z);
1790   8                             END;
1791   7                        IF EQUAL(.('M'),.PHRASE,1)
                                   THEN DO;
1793   8                             FIELD(16), FIELD(17), FIELD(18) = 0FFH;
1794   8                             CALL SETBIT(.MWORD,9,3,Z);
1795   8                             END;
1796   7                        IF EQUAL(.('S'),.PHRASE,1)
                                   THEN DO;
1798   8                             FIELD(11), FIELD(12) = 0FFH;
1799   8                             CALL SETBIT(.MWORD,18,2,Z);
1800   8                             END;
1801   7                        END;
1802   6                END;
```

```
                $EJECT

1803   5                END;    /* OF DO CASE TYPE */
1804   4                END;    /* OF DO WHILE PHRASE(0)<>';' */
```

```
$EJECT

                    /* DELAYED RESOLUTION OF S1 AND A BITS */

1805   3            DO;
1806   4            DECLARE (INDEX, S1, A, TBLENTRY) BYTE;

                   /* COMPOSE TABLE INDEX */
1807   4            INDEX = 0;
1808   4            IF P2BREGPHRASEPRESENT     THEN INDEX = INDEX OR 80H;
1810   4            IF NOTAD1                  THEN INDEX = INDEX OR 40H;
1812   4            IF FIELD(11)               THEN INDEX = INDEX OR 20H;
1814   4            IF EQUBIT(.MWORD,18,1,1B)  THEN INDEX = INDEX OR 10H;
1816   4            IF FIELD(7)                THEN INDEX = INDEX OR 08H;
1818   4            IF FIELD(8)                THEN INDEX = INDEX OR 04H;
1820   4            IF EQUBIT(.MWORD,16,1,1B)  THEN INDEX = INDEX OR 02H;
1822   4            IF EQUBIT(.MWORD,17,1,1B)  THEN INDEX = INDEX OR 01H;

                   /* FETCH TABLE ENTRY */
1824   4            TBLENTRY = TBL(INDEX);

                   /* DECODE TABLE ENTRY */
1825   4            IF (TBLENTRY AND 80H)<>0
                       THEN DO;
1827   5                      E = E + 1;
1828   5                      CALL ALPHA(LP,47,.('*** UNFORSEEN CIRCUMSTANCE ',
                                   'HAS ARISEN.  INDEX = '));
1829   5                      CALL HEX(LP,2,.INDEX);
1830   5                      CALL EOL;
1831   5                      GOTO ENDRESOLVE;
1832   5                      END;
1833   4            IF (TBLENTRY AND 08H)<>0
                       THEN DO;
1835   5                      E = E + 1;
1836   5                      CALL ALPHA(LP,61,.('*** SPECIFICATION FOR A AND ',
                                   'S1 CONFLICT.  MICROWORD NOT VALID.'));
1837   5                      CALL EOL;
1838   5                      END;
1839   4                  ELSE DO;
1840   5                      S1 = SHR(TBLENTRY,2) AND 01H;
1841   5                      CALL SETBIT(.MWORD,18,1,S1);
1842   5                      FIELD(11) = 0FFH;
1843   5                      A = TBLENTRY AND 03H;
1844   5                      CALL SETBIT(.MWORD,16,2,A);
1845   5                      FIELD(7), FIELD(8) = 0FFH;
1846   5                      END;
1847   4      ENDRESOLVE:END;
```

```
                $EJECT

                        /* DELAYED RESOLUTION OF M BITS.  */

1848    3       IF DOALCPHRASEPRESENT
                   THEN DO;
1850    4                IF FIELD(16) AND EQUBIT(.MWORD,9,1,1B)
                           THEN DO;
1852    5                      IF FIELD(17) AND NOT EQUBIT(.MWORD,10,1,0B)
                                 THEN DO;
1854    6                             E = E + 1;
1855    6                             CALL ALPHA(LP,57,.('*** CONFLICTING ',
                                      'SPECIFICATION FOR M.  MICROWORD ',
                                      'NOT VALID.'));
1856    6                             CALL EOL;
1857    6                             END;
1858    5                           ELSE DO;
1859    6                             CALL SETBIT(.MWORD,10,1,0B);
1860    6                             FIELD(17) = 0FFH;
1861    6                             IF NOT FIELD(18)
                                         THEN DO;
1863    7                                     FIELD(18) = 0FFH;
1864    7                                     CALL SETBIT(.MWORD,11,1,1B);
1865    7                                     END;
1866    6                             END;
1867    5                      END;
1868    4                  ELSE DO;
1869    5                      CALL SETBIT(.MWORD,9,1,0B);
1870    5                      FIELD(16) = 0FFH;
1871    5                      END;
1872    4                END;

1873    3       IF P1BDIPHRASEPRESENT
                   THEN DO;
1875    4                IF EQUBIT(.MWORD,9,3,000B)
                            THEN DO;
1877    5                      IF FIELD(16) AND FIELD(17) AND FIELD(18)
                                 THEN DO;
1879    6                             E = E + 1;
1880    6                             CALL ALPHA(LP,57,.('*** CONFLICTING ',
                                      'SPECIFICATION FOR M.  MICROWORD ',
                                      'NOT VALID.'));
1881    6                             CALL EOL;
1882    6                             END;
1883    5                           ELSE DO;
                                      /* SINCE M2=1 BY DEFAULT,          */
                                      /* FIELD(16) MUST BE SET IF M=0.  */
1884    6                             IF NOT FIELD(17)
                                         THEN DO;
1886    7                                     FIELD(17) = 0FFH;
1887    7                                     CALL SETBIT(.MWORD,10,1,1B);
1888    7                                     END;
1889    6                                 ELSE DO;
                                              /* FIELD(18) IS NOT SET */
1890    7                                     FIELD(18) = 0FFH;
1891    7                                     CALL SETBIT(.MWORD,11,1,1B);
1892    7                                     END;
```

```
1893   6                          END;
1894   5                    END;
1895   4              END;
```

```
                $EJECT

1896   3                CALL PRINTROUTINE;
1897   3        DECODED:END; /* OF DO WHILE 0FF */
1898   2                END PASS2;
1899   1                END MICRO2;


MODULE INFORMATION:

     CODE AREA SIZE      = 4CCEH  19662D
     VARIABLE AREA SIZE  = 02D8H    728D
     MAXIMUM STACK SIZE  = 000CH     12D
     2466 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

5.1.2 Q-80 Simulator

5.1.2.1 User's Reference

Inputs

The Q-80 Simulator reads input from 3 disk files:

- the object code file,
- the schedule file, and
- the data input file.

The object file contains object code produced by the Intel 8080/8085 Macro Assembler V2.0 (an Intel product). The schedule file specifies the behavior of the HALT, HOLD, INT and RESIN inputs of the Q-80 during the execution of the program contained in the object file. The 3 bytes to be jammed in case of interrupt are also specified in the schedule file. This file may be omitted. The data input file specifies a stream of bytes that is read whenever a Q-80 IN-instruction is executed. This file may also be omitted.

Function

The Q-80 simulator is implemented in principle as a finite automaton and the possible state transitions are shown in Figure 5.1-13. The state transitions are determined by input read from the object file (opcodes) and by input read from the schedule file (HALT, HOLD, INT, RESET, opcodes of instructions jammed at interrupt). For each state transistion one 6-byte record is written on the disk. For each state visited a line of printed output is generated. Simulated execution ends when an HLT-instruction is executed and none of the following is specified:

HALT=1, HOLD=1, INT=1 or RESET=1.

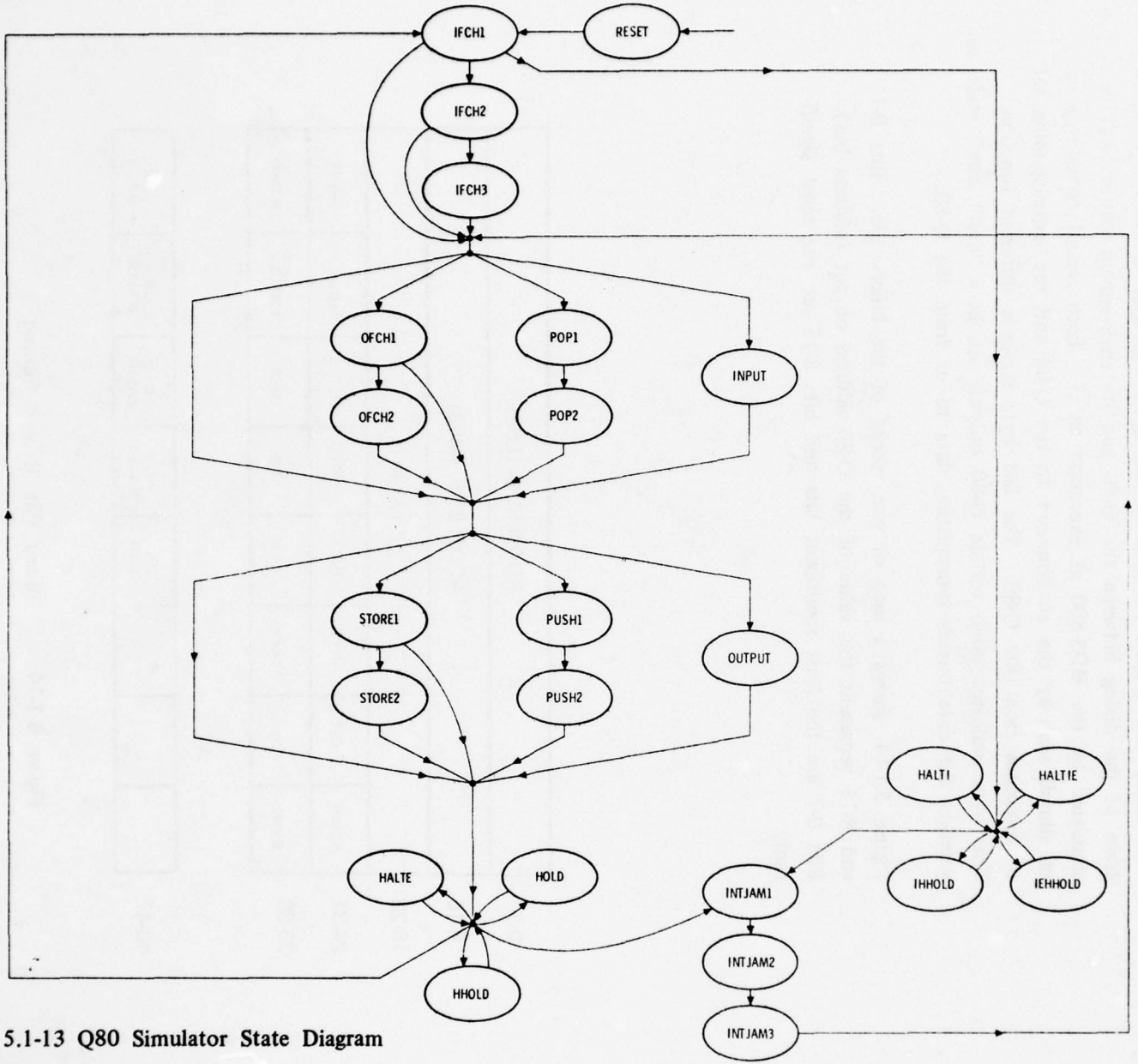


Figure 5.1-13 Q80 Simulator State Diagram

Output

The binary file created on disk by the Q-80 simulator contains a complete trace of the dialog between the Q-80 and its environment (which will be simulated by the MDS-800 at execution time). Each record contains a set of stimuli sent by the environment to the Q-80 and the corresponding set of responses from the Q-80. The last byte serves a different function. This byte indicates when certain Q-80 outputs are in a "don't care" state and whether the data bus is transmitting data to or from the Q-80.

Figure 5.1-14 shows a map of one record of the binary file. Bits 0-7 and 8-15 represent the value of the Q-80 address output (address bus). Bits 0-7 are the least significant bits and bits 8-15 are the most significant.

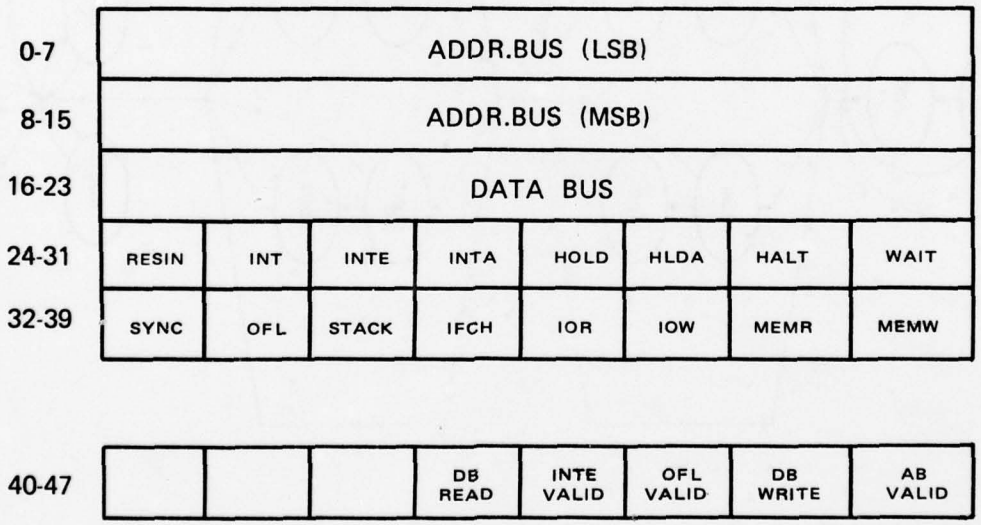


Figure 5.1-14 Binary File Record Format

Bits 16-23 represent the value on the data bus (which may be a stimulus or a response since this bus is bidirectional). Bits 24, 25, 28 and 30 are the stimuli RESIN, INT, HOLD and HALT, respectively. Bits 26, 27, 29, 31, 33 are the responses INTE, INTA, HLDA, WAIT and OFL, respectively. Bits 34-39 are the memory and I/O status signals STACK, IFCH, IOR IOW, MEMR and MEMW. Bits 43 and 46 indicate when the data bus is used for a read or a write operation, respectively. If both bits are 0 the value on the bus is of no consequence (don't care). Similarly, if bit 47 is 0, the value on the address bus is of no consequence. Bits 44 and 45 indicate whether INTE and OFL have yet been initialized (i.e. 0 means don't care).

The listing created by the Q-80 Simulator also represents a complete trace of the dialog between the Q-80 and its environment. Here each line represents a set of responses from the Q-80 and the next set of stimuli sent to the Q-80. This is different from the definition of a record in the binary file and thus individual lines of the listing are not in a one-to-one correspondence to individual records in the binary file. These two different approaches were necessary for the following reason. The disk file is to be used by an exerciser program to simulate the Q-80's environment and check its proper operation. Such an exerciser applies a set of stimuli, then checks if the Q-80's response was as expected. This leads to the address part of a memory access and the data read to be in two consecutive records. A listing organized along the same principles is hard to follow. For that reason, stimuli and responses are paired for the listing such that the memory request (a Q-80 response) and the memory response (a Q-80 stimulus) are on the same line.

Date File

A file called :F0:DATE is read by the Q-80 Simulator. The first 11 bytes in this file are assumed to be the current date. The file must be edited daily by the user. The format of the date is shown in the following example:

12 FEB 1978.

Calling

To call the Q-80 Simulator from ISIS-II (which prompts with a –) enter:

Q80SIM :F1:name.OBJ

on the keyboard. This implies that :F1:name.OBJ is the object file, that :F1:name.SCH is the schedule file (may not exist), that :F1:name.IN is the input file (may not exist) and that :F1:name.BIN will be the binary file.

States and state transitions.

The Q-80 Simulator has 24 states. Each of these states corresponds to a wait-loop in the microprogram of the Q-80.

Transitions between states are caused by the arrival of new stimuli from the environment of the Q-80. A state transition corresponds to the execution of microcode between two wait-loops. It begins with the exit from one loop, continues with the execution of non-iterative code and ends with entering another (prehaps the same) loop.

- **The state RESET**

This state is entered from any state if RESIN=1. Every simulation run starts out in this state.

The program counter is set to 0 and the address and data buses are floated. Depending on the value of the stimulus RESIN, the next state is RESET or IFCH1.

- **The state IFCH1.**

In this state the opcode of the next instruction to be executed is fetched from memory. The value of the program counter is placed on the address

bus and IFCH and MEMR are set to 1. After the opcode is received, the program counter is incremented and OFL is cleared.

If the opcode fetched indicates that the instruction is a 2 or 3 byte instruction the next state is IFCH2. If it is a 1-byte instruction which requires an operand to be popped from the stack, the next state is POP1 or COND*. If it is an instruction which requires an operand to be fetched from memory (other than pop from the stack) the next state is OFCH1. If the instruction is the IN-instruction, the next state is INPUT. If it is a 1-byte instruction which does not require any sort of operand fetch, the instruction is executed at this point by a transition to the pseudo-state EXEC.

- **The state IFCH2.**

In this state the 2nd byte of a 2 or 3-byte instruction is fetched from memory. The value of the program counter is placed on the address bus and IFCH and MEMR are set to 1. After the 2nd byte is received, the program counter is incremented.

If the opcode, fetched in IFCH1, indicates that the instruction is 3 bytes long the next state is IFCH3. Otherwise the next state is EXEC, OFCH1, POP1 or INPUT depending on the opcode (as in state IFCH1).

- **The state IFCH3.**

In this state the 3rd byte of a 3-byte instruction is fetched from memory. The value of the program counter is placed on the address bus and IFCH and MEMR are set to 1. After the 3rd byte is received, the program counter is incremented. The next state is EXEC, OFCH1 or POP1. depending on the opcode (as in state IFCH1).

---

* The 8 conditional return instructions are a special case. The condition must be examined first, to determine whether to pop the stack. This is done in the pseudo-state COND.

- **The pseudo-state COND**

This pseudo-state does not cause output to the binary file or to the listing. It is only an intermediate step in the execution of conditional return instructions which serves to decide the next state after IFCH1. If the condition is true, the next state is POP1, otherwise it is the pseudo-state EXEC.

- **The state OFCH1.**

In this state single-byte operands and the 1st bytes of 2-byte operands are fetched. Depending on the opcode one of the following is placed on the address bus:

- the register pair BC
- the register pair DE
- bytes 2 and 3 of the instruction
- the register pair HL,

and MEMR is set to 1. Depending on the opcode, the next state is either OFCH2 (2-byte operands) or EXEC (1-byte operands).

- **The state OFCH2.**

In this state the 2nd byte of a 2-byte operand is fetched. The address used in OFCH1 is incremented and placed on the address bus. MEMR is set to 1. The next state is always the pseudo-state EXEC.

- **The state POP1.**

In this state the 1st of 2 bytes (the LSB) are popped from the stack. The stack pointer (SP) is placed on the address bus and STACK and MEMR are set to 1. After the byte is received from memory the stack pointer is incremented. The next state is always POP2.

- **The state POP2.**

In this state the 2nd of 2 bytes (the MSB) is popped from the stack. As in POP1, the stack pointer is placed on the address bus and STACK and MEMR are set to 1. After the byte is received from memory the stack pointer is incremented. The next state is always the pseudo-state EXEC.

- **The state INPUT.**

In this state a byte is read from the port indicated by the 2nd byte of the instruction. The port number (2nd byte of the instruction) is placed on the address bus, padded with leading zeros. IOR is set to 1. The Q-80 simulator will supply the input byte, reading the next byte from the file :F1:name.IN. If the file does not exist or if the end of file has been reached the simulation is aborted. The next state is always the pseudo-state EXEC.

- **The pseudo-state EXEC**

The pseudo-state EXEC represents the execution phase of the 256 Q-80 instructions. Direct transitions from the states IFCH1, IFCH2, IFCH3, OFCH1, OFCH2, POP2, and INPUT to the states STORE1, PUSH1, OUTPUT, HALTE, HOLD, HHOLD, HALTI, HALTIE, IHHOLD, IEHHOLD, INTJAM1 and IFCH1 all pass through this pseudo-state. The pseudo-state. The pseudo-state EXEC does not cause any output to the binary file or to the listing. The next instruction is determined as follows. If the opcode indicates that 2 bytes must be pushed onto the stack then the next state is PUSH1. If a 1 or 2-byte result must be stored in the memory (not on the stack) the next state is STORE1. If the instruction is the OUT-instruction the next state is OUTPUT. If the instruction is an HLT-instruction the next state is one of HALTI, HALTIE, IHHOLD, IEHHOLD or INTJAM1 (see Table 5.1-8).

Table 5.1-8. Next state at end of execution for HLT-instructions

HALT	HOLD	INT.INTE	NEXT STATE
0	0	0	HALTI
0	0	1	INTJAM1
0	1	X	IHHOLD
1	0	X	HALTIE
1	1	X	IEHHOLD

For all other instructions the next state is one of IFCH1, INTJAM1, HALTE, HOLD or HHOLD (see table 5.1-9).

Table 5.1-9. State transition at end of instruction execution (except HLT)

HALT	HOLD	INT.INTE	NEXT STATE
0	0	0	IFCH1
0	0	1	INTJAM1
0	1	X	HOLD
1	0	X	HALTE
1	1	X	HHOLD

AD-A063 004 QUESTRON CORP SAN DIEGO CALIF F/G 9/2
RADIATION HARDENED MICROPROCESSOR. VOLUME II.(U)
MAY 78 V V NICKEL, P A ROSENBERG F33615-77-C-1001
UNCLASSIFIED AFAL-TR-78-55-VOL-2 NL

3 OF 4
AD A063 004

- **The state STORE1.**

In this state single-byte results and the 1st byte of 2-byte results are stored stored in the memory. Depending on the opcode one of the following is placed on the address bus:

- the register pair BC
- the register pair DE
- bytes 2 and 3 of the instruction
- the register pair HL.

The byte to be stored is placed on the data bus. MEMW is set to 1. For 1-byte results the next state is one of IFCH1, INTJAM1, HALTE, HOLD or HHOLD depending on the value of HALT, HOLD and of INT.INTE (see Table 5.1-9). For 2-byte results the next state is STORE2.

- **The state STORE2.**

In this state the 2nd byte of 2-byte results is stored in memory. The address used in STORE1 is incremented and placed on the address bus. The byte to be stored is placed on the data bus. MEMW is set to 1. The next state is one of IFCH1, INTJAM1, HALTE, HOLD, or HHOLD (see Table 5.1-9).

- **The state PUSH1.**

In this state the 1st of 2 bytes (the MSB) is pushed onto the stack. The stack pointer (SP) is decremented first and the decremented value is placed on the address bus. The byte to be pushed is placed on the data bus. STACK and MEMW are set to 1. The next state is always PUSH2.

- **The state PUSH2.**

In this state the 2nd of 2 bytes (the LSB) is pushed onto the stack. The stack pointer (SP) is decremented first and the decremented value is placed on the address bus. The byte to be written into memory is placed on the

data bus. STACK and MEMW are set to 1. The next state is one of IFCH1. INTJAM1, HALTE, HOLD or HHOLD (see Table 5.1-9).

- **The state OUTPUT**

In this state a byte is written to the port indicated by the 2nd byte of the instruction. The port number (2nd byte of the instruction) is placed on the address bus, padded with leading zeros. The byte to be written is placed on the data bus. IOW is set to 1. The next state is one of IFCH1, INTJAM1, HALTE, HOLD or HHOLD (see Table 5.1-9).

- **The state HALTE**

This state is entered at the end of the execution of any instruction other than HLT if HALT=1 and HOLD=0. WAIT is set to 1. The next state is one of IFCH1, INTJAM1, HALTE, HOLD or HHOLD (see Table 5.1-9).

- **The state HOLD.**

This state is entered at the end of the execution of any instruction other than HLT if HOLD=1 and HALT=0. The address bus and the data bus are floated and HLDA is set to 1. The next state is one of IFCH1, INTJAM1, HALTE, HOLD or HHOLD (see Table 5.1-9).

- **The state HHOLD**

This state is entered at the end of the execution of any instruction other than HLT if HALT=1 and HOLD=1. The address bus and the data bus are floated. HLDA and WAIT are set to 1. The next state is one of IFCH1, INTJAM1, HALTE, HOLD or HHOLD (see Table 5.1-9).

- **The state HALTI.**

This state is entered at the end of an HLT-instruction if HALT=0 and HOLD=0. WAIT is set to 1. The next state is one of HALTI, HALTIE, IHHOLD, IEHHOLD or INTJAM1 (see Table 5.1-8).

- **The state HALTIE.**

This state is entered at the end of an HLT-instruction if HALT=1 and HOLD=0. WAIT is set to 1. The next state is one of HALTI, HALTIE, IHHOLD, IEHHOLD or INTJAM1 (see Table 5.1-8).

- **The state IHHOLD.**

This state is entered at the end of an HLT-instruction if HOLD=1 and HALT=0. HLDA and WAIT are set to 1 and both the address bus and the data bus are floated. The next state is one of HALTI, HALTIE, IHHOLD, IEHHOLD or INTJAM1 (see Table 5.1-8).

- **The state IEHHOLD.**

This state is entered at the end of an HLT-instruction if HALT=1 and HOLD=1. HLDA and WAIT are set to 1 and both the address and data buses are floated. The next state is one of HALTI, HALTE, IHHOLD, IEHHOLD or INTJAM1 (see Table 5.1-8).

- **The state INTJAM1.**

In this state the first of 3 bytes (opcode) is jammed into the Q-80 as a result of the recognition of an interrupt. Further interrupts are disabled by setting INTE to 0. The value of the program counter is placed on the address bus and INTA and IFCH are set to 1. After the first jammed byte is received, OFL is set to 0. The next state is IFCH2.

- **The state IFCH2**

In this state the 2nd of 3 bytes is jammed into the Q-80 as a result of the recognition of an interrupt. The value of the program counter is placed on the address bus and INTA and IFCH are set to 1. The next state is INTJAM3.

- **The state INTJAM3.**

In this state the 3rd of 3 bytes is jammed into the Q-80 as a result of the recognition of an interrupt. The value of the program counter is placed on the address bus and INTA and IFCH are set to 1. The next state is one of EXEC, OFCH1, POP1 or INPUT depending on the opcode (first byte jammed).

Format of the Schedule File

The value of the Q-80 stimuli RESIN, HALT, HOLD and INT may be specified by means of the schedule file. The schedule file is normally produced with the aid of the Intel ISIS-II Text Editor (an Intel product) and is thus an ASCII file. Each line in the this file refers to a specific instance of the execution of a specific instruction. The first four characters on each line are interpreted as hexadecimal digits (ASCII 0-9, A-F) and specify the memory address where the opcode of the instruction is located. These four digits are followed by a TAB (control-I) character. The next 1-5 characters represent the count, a decimal number between 1 and 65536. If the count is 1, the line refers to the first time the specified instruction is executed. If it is 2, it refers tothe 2nd time the instruction is executed, etc., etc. The count field is terminated by a TAB (control-I). For example:

03DC (TAB) 35 (TAB) . . . . . .

indicates that this line refers to the 35th time the instruction located at address

The next field specifies the value of the stimuli RESIN, HALT, HOLD and INT (RHHI for short). There are two ways to specify the value of RHHI. If the specification refers to one of the states listed below, the format is:

<RHHI> @ <state>

where <RHHI> is a string of four 0-s and 1-s and where <state> is the name of the state in which the stimulus is to be applied. For example:

1000@OFCH2

indicates that RESIN=1, HALT=0, HOLD=0, INT=0 are the stimuli to be applied in state OFCH2 of the instruction execution specified. Specifications of the format described above may be used only for the following states:

IFCH1, IFCH2, IFCH3

OFCH1, OFCH2

POP1, POP2

INPUT

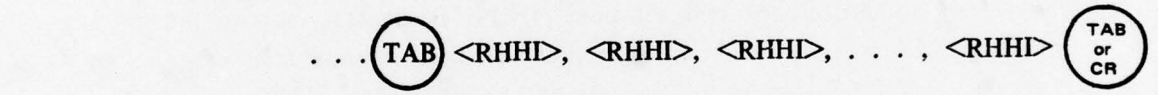
STORE1, STORE2

PUSH1, PUSH2

OUTPUT

INTJAM1, INTJAM2, INTJAM3.

For the remaining states RHHI-stimulus values are specified by use of the second format. These are states entered at the end of the execution of an instruction and sequences of state transitions of arbitrary length may occur before the execution of the next instruction commences. For this reason a sequence of up to 8 RHHI-values can be specified for an instruction execution:



Note that no state is specified. Application of stimuli from this sequence begins as soon as one of the states

RESET

HALTE, HOLD, HHOLD

HALTI, HALTIE, IHHOLD, IEHHOLD

is entered and continues until a transition to either IFCH1 or INTJAM1 is made. If the end of the sequence is reached the next RHHI-stimulus will be 0000 which will cause a transition to IFCH1 or ends the simulation run. (The reader will recall that the simulation run ends if RHHI=0000 in state HALTI.)

The following are examples of valid lines in a schedule file:

03DC (TAB) 35 (TAB) 1000@OFCH2 (CR)

20A3 (TAB) 16 (TAB) 01000, 0100, 0110, 0010 (CR)

If an RHHI-stimulus specifies INT=1, the RHHI-specification field of the respective line may be followed by another TAB (control-I), the word JAM=, and six hexadecimal digits, which specify the 3 bytes to be jammed, and a CR (carriage return). For example:

2794 (TAB) 2 (TAB) 0001@STORE1 (TAB) JAM=C30008 (CR) .

The schedule file may contain at most 50 lines. No blanks may occur anywhere in the file. The Q-80 Simulator echoes the lines read from the schedule file on the listing. Any errors in the schedule file cause aborting of the simulation run. For any state of the simulation run where none of the lines applies, RHHI=0000 is assumed. If there is no schedule file, i.e. :F1:name.SCH does not exist, RHHI=0000 is assumed at all times.

5.1.2.2 Q-80 Simulator Computer Program Documentation*

Environment

The Q-80 Simulator is written in the language PL/M-80 and is designed to run under the ISIS-II operating system (PL/M-80 V3.0 and ISIS-II V2.2 are Intel products). In addition to the features of PL/M-80, a number of external library routines are used as primitives in the program. A brief functional description of these routines follows.

OPEN -- This subroutine opens a file. Given a pointer to the filename and an access code (1=read, 2=write, 3=read & write) this subroutine returns an index into the Active File Table (AFT) to be used when accessing the file and a status word.

Argument 1 = pointer to location where AFT index is to returned
Argument 2 = pointer to filename
Argument 3 = access code
Argument 4 = AFT index of echo file or 0 if no echo
Argument 5 = pointer to location where status word is to be returned.

This subroutine is declared in lines 2-4 of the source listing.

CLOSE -- This subroutine closes a file.

Argument 1 = AFT index of file
Argument 2 = pointer to location where status word is to be returned.

This subroutine is declared in lines 5-7 of the source listing.

READ -- This subroutine reads from a file. Given the AFT index of a file, a pointer to the read buffer and the number of bytes to be read, this subroutine will fill the buffer, return the number of bytes actually read and the status word.

---

* Familiarity with the ISIS-II operating system and PL/M-80 compiler is assumed. It is also assumed that the reader has read the previous section describing the operation and use of the Q-80 Simulator.

Argument 1 = AFT index of file

Argument 2 = pointer to buffer

Argument 3 = number of bytes to be read

Argument 4 = pointer of location where actual number of bytes read is to be returned

Argument 5 = pointer to location where status word is to be returned.

This subroutine is declared in lines 8-10 of the source listing.

WRITE – This subroutine writes into a file. Given the AFT index of the file, a pointer to the write buffer and the number of bytes to be written, this subroutine will write the buffer contents into the file and return the status word.

Argument 1 = AFT index of file

Argument 2 = pointer to buffer

Argument 3 = number of bytes to be written

Argument 4 = pointer to location where status word is to be returned.

This subroutine is declared in lines 11-13 of the source listing.

SEEK – This subroutine allows the repositioning of the file pointer, i.e. facilitates skipping around in the file. Many modes allow a variety of ways to move the file pointer, but some modes do not work correctly (Intel will fix in next release of ISIS-II). For this reason only two modes (modes 0 and 2) are used in this program. In mode 0 SEEK returns the block number and byte number where the filepointer is pointing (1 block = 128 bytes). In mode 2 the filepointer is set based on a blocknumber – bytenumber pair supplied as arguments.

Argument 1 = AFT index of file

Argument 2 = mode (0 or 2)

Argument 3 = pointer to blocknumber

Argument 4 = pointer to bytenumber

Argument 5 = pointer to status word

This subroutine is declared in lines 14-16 of the source listings.

DELETE – This subroutine deletes a file.

Argument 1 = AFT index of file to be deleted

Argument 2 = pointer to status word.

This subroutine is declared in lines 17-19 of the source listing.

ERROR – This subroutine displays a standard error message on the CRT. It is to be used when an abnormal status word was returned by another subroutine.

Argument 1 = error number

Argument 2 = pointer to status word.

This subroutine is declared in lines 20-22 of the source listing.

EXIT – This subroutine terminates a program and returns to ISIS-II. There are no arguments. This subroutine is declared on lines 23-24 of the source listing.

ALPHA – Simplified version of WRITE. It is mostly used to display alpha-numeric data on the CRT or on the printer.

Argument 1 = AFT index of file

Argument 2 = number of bytes to be written

Argument 3 = pointer to buffer.

This subroutine is declared in lines 25-27 of the source listing.

HEX – Converts the contents of a buffer to its hexadecimal representation in ASCII and writes those characters into the file indicated (mostly CRT or LP).

Argument 1 = AFT index of file

Argument 2 = number of hexadecimal digits to be displayed

Argument 3 = pointer to buffer.

This subroutine is declared in lines 28-30 of the source listing.

INTEG – This subroutine converts a byte (considered as an 8 bit 2's complement interger) to signed decimal in ASCII and writes those characters into the file indicated (mostly CRT or LP).

Argument 1 = AFT index of file

Argument 2 = data (1 byte).

This subroutine is declared in lines 31-33 of the source listing.

INTEGD – This subroutine converts a two-byte unsigned binary number (type ADDRESS) to its 5-digit decimal equivalent in ASCII and writes those characters into the file indicated (mostly CRT or LP).

Argument 1 = AFT index of file

Argument 2 = data (2 bytes)

This subroutine is declared in lines 34-36 of the source listing.

PAGE – Sends the form-feed character (hexadecimal 0C) to the file indicated. If the file is the printer, the paper is advanced to the top of the next page. The only argument is the AFT index of the file. This subroutine is declared on lines 37-39 of the source listing.

PPAGE – Like PAGE, except the vertical-tab character (hexadecimal 0B) is sent. The Centronics 306 printer equipped with a vertical format unit (VFU) can be programmed to skip in response to this character to either the first or second top of page, such that the face of first page folds on top of the listing (not the back of the first page). This subroutine is declared in lines 40-42 of the source listing.

SKIP – Sends a carriage-return and a line-feed character (0D0A) to the file specified. The only argument is the AFT index of the file. This subroutine is declared in lines 43-45 of the source listing.

SPACE – This subroutine sends a specified number of blanks to the file indicated.

Argument 1 = AFT index of file

Argument 2 = number of blanks.

This subroutine is declared in lines 46-48 of the source listing.

EQUAL – This function compares two given bit strings and returns the value FF if they are equal, or 00 if they are not.

Argument 1 = pointer to first string

Argument 2 = pointer to second string

Argument 3 = number of bytes to be compared

This function is declared in lines 49-51 of the source listing.

SETBIT – This subroutine can set up to 7 bits in a bit string.

Argument 1 = pointer to string

Argument 2 = start-bit number (0-origin)

Argument 3 = number of bits to be set

Argument 4 = data (1 byte). The least significant bits are used to change the bit string.

This subroutine is declared in lines 52-54 of the source listing.

EQUBIT – This function compares a given bit string to a substring of a longer bit string and returns the value FF if they are equal, or 00 if they are not.

Argument 1 = pointer to ling bit string

Argument 2 = start-bit number (0-origin)

Argument 3 = number of bits to be compared

Argument 4 = short bit string (1 byte, number of least significant bits to be compared is indicated by argument 3).

This subroutine is declared in lines 55-57 of the source listing.

SUBSTR – This function returns part of a given bit string.

Argument 1 = pinter to given string

Argument 2 = start-bit number (0-origin) where substring begins

Argument 3 = length of substring

The substring is returned right justified. This subroutine is declared in lines 58-60 of the source listing.

The computer program description, which follows, describes a program with deep nesting of blocks and loops. Accordingly, the discussion begins at the highest level and proceeds to deeper levels of nesting and at the same time to levels of increasing details. To make the relationship of the various sections that follow as clear as possible, the sections are numbered like the Dewey Decimal System. Thus for a dection numbered 3.1.8, further details can be found in sections 3.1.8.1, 3.1.8.2, . . . etc., to find the relationship of section 3.1.8 to sections 3.1.1, 3.1.2, . . . etc., read section 3.1.

The main program consisting of the outermost block is labelled Q80SIM and represents the global environment. It contains

1.) Declaration of external global procedures used as primitives in the remainder of the program (described above, lines 2-60).

2.) Declaration of global constants and variables (lines 61-129).

3.) Declaration of internal global constants and variables (lines 69-129).

4.) The "Prologue Activities" block (lines 130-170).

5.) The "Read Object File" block (lines 171-200).

6.) The "Read Schedule File" block (lines 201-393).

7.) The "Q-80 Simulation" block (lines 394-2700).

5.1.2.2.1 Declaration of external global procedures.

These procedures were described in detail above.

5.1.2.2.2 Declaration of global constants and variables.

Global constants and variables are declared in lines 61-68. These global declarations fall into the following categories:

- AFT-indexes for the object file (BI), the binary output file (BO), the line printer (LP), the Q-80 input file (Q80IN), the schedule file (SCH), the console keyboard (KB) and the console CRT (CRT).

- Special ASCII character constants for carriage return (CR) and tab (TAB).

- Paging counters: PAGENO and LINENO.

- An array holding the name of the object file (FN).

- The structure COMM holding the information read from the schedule file in compact internal format.

- The variable INSTATUS which indicates the existence of the file :F1:name.SCH.

5.1.2.2.3 Declaration of internal global procedures.

There are two global internal procedure both related to page formatting of the output listing: HEADER and EOL (which stands for end-of-line).

Subroutine header (lines 69-122) has one parameter, SW, which should be 1 when the first page is printed, and 0 for all other calls of HEADER. Figure 5.1.2-4 shows two headers produces by calls of subroutine HEADER with SW=1 and SW=0 respectively.

Subroutine EOL (lines 123-129) is called anytime a line is ended. It increments LINENO and sends a carriage return character to the line printer. If LINENO exceeds 61, HEADER is called from EOL.

5.1.2.2.4 The "Prologue Activities" Block.

This block is located in lines 130-170. In this block the object file, the Q-80 input file and the line printer are opened.

First, the remainder of the command line, entered on the keyboard when Q80SIM was called from ISIS-II, is read into array LINE. After skipping over leading blanks, only the characters forming the name of the object file are copied to array FN and any trailing characters are filled with blanks.

Next, the object file is opened. If the file can not be opened (STATUS≠0) the message "*** CAN NOT OPEN FILE. STATUS=xxx" is displayed on the CRT and the simulation run is aborted. If the object file was successfully opened, the line printer is opened next, PAGENO is set to 1 and HEADER is called to print the first page header (argument=1).

The 3-character file extension of the file name in FN is changed from .OBJ to .BIN and then to .IN as the binary output file and the Q-80 input file are opened. The global variable INSTATUS is set to indicate the existence of the Q-80 input file (INSTATUS=13 indicates that the file does not exist).

5.1.2.2.5 The "Read Object File" Block.

This block is located in lines 171-200. The contents of the object file are loaded into the array MEMORY which is located between the end of the program and the physical end of the MDS-800 memory. The size of this array may vary as future changes are incorporated into the Q-80 Simulator. Currently with 64K bytes of memory in the MDS-800, over 19K bytes are available for simulated Q-80 memory.

The object files produced by the Intel 8080/8085 Macro Assembler are organized in blocks (Figure 5.1-15). The first byte identifies the type of block in question. Only blocks where this byte contains the value 6 are object code blocks. The next two bytes indicate the length of the block (see Figure 5.1-15 for a more precise understanding of the meaning of these two bytes). The byte following the length is of no

interest and the next two bytes indicate the address where the 1st true object code byte of this block is to be loaded. The next "length"-3 bytes are true object code.

The "Read-Object File" block reads files of the above described organization into array MEMORY. Any attempt to write past the end of array MEMORY is trapped and the simulation run is aborted after printing the message: "*** ATTEMPT TO LOAD OUT OF BOUNDS MEMORY LOCATION." When the end of the object file is reached, the file is closed.

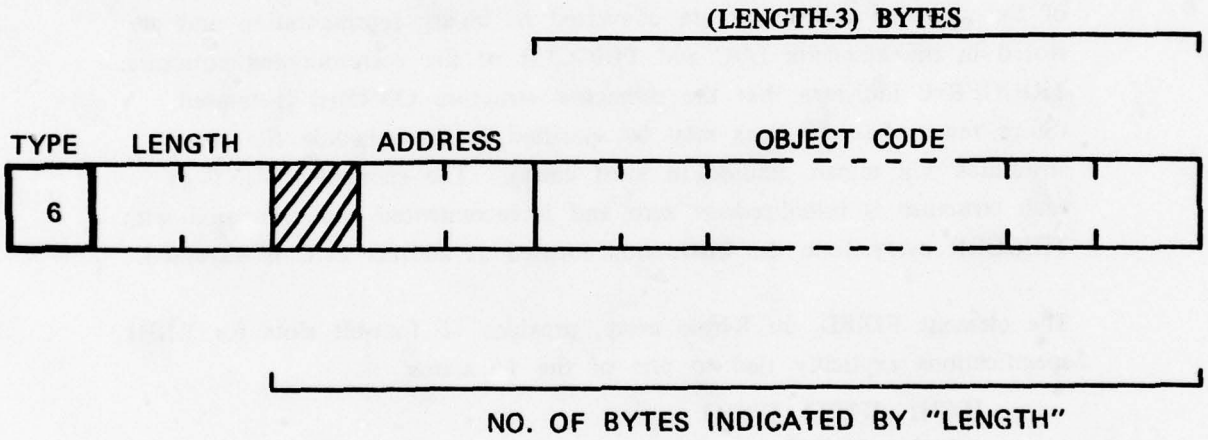


Figure 5.1-15. Format of blocks in object code file.

5.1.2.2.6 The "Read Schedule File" Block.

This block is located in lines 201-393. The function HEX, which converts an ASCII hexadecimal 'digit to the corresponding binary value, is nested within this block, at lines 208-219. If the argument of HEX is not a hexadecimal digit the simulation run is aborted after displaying the message "*** NON-HEX DIGIT." on the CRT.

The schedule file :F1:name.SCH must be opened first. The 3-character file extension of the filename in array FN is changed to .SCH. The schedule file is opened in line-edit mode with no echo (i.e. the bit-bucket file :BB: is opened and assigned as echo file).

The structure COMM (global, declared line 67) is an array of 50 structures. Each line in the schedule file (max. 50 lines) corresponds to one of these structures (COMM(0) through COMM(49)).* The hexadecimal address and the decimal number which specifies the particular instance of execution of the specified instruction are converted to binary representation and are stored in the elements LOC and TRIGGER of the corresponding structure. TRIGGER=0 indicates that the respective structure COMM(i) is unused. (Since fewer than 50 lines may be specified in the schedule file some structures will remain unused in most cases). The element COUNT in each structure is initialized to zero and is incremented and compared with TRIGGER every time the instruction located at address LOC is executed.

The element FIXED, an 8-byte array, provides 16 four-bit slots for RHHI specifications explicitly tied to one of the 16 states:

IFCH1, IFCH2, IFCH3
OFCH1, OFCH2
POP1, POP2
INPUT
STORE1, STORE2
PUSH1, PUSH2
OUTPUT
INTJAM1, INTJAM2, INTJAM3

by specifications of the form

<RHHI>@<STATE>

---

* It may be helpful if the reader would review the subsection titled "Format of the schedule file", in section 5.1.2.1, at this point.

The element SEQU, an array of 4 bytes, can hold up to 8 RHHI specifications to be used at the end of the execution of instructions to specify transitions from the states:

HALTE, HOLD, HHOLD
HALTI, HALTIE, IHHOLD, IEHHOLD.

The element JAM, an array of 3 bytes, holds the 3 bytes to be jammed in case of interrupt.

Thus the remainder of the "Read Schedule File" block is dedicated to translating the contents of the schedule file to the format of the structure file to the format of the structure COMM. If any syntax error is detected the simulation run is aborted. All lines read from the schedule file are displayed on the CRT and printed on the line printer. Error messages are displayed on the CRT. When the end of the schedule file is reached the file is closed.

5.1.2.2.7 The "Q-80 Simulation" Block.

Starting with line 394, the remainder of the program is taken up by the "*Q-80 Simulation*" *block. This block contains the following:*

1.) Declaration of variables and tables.
2.) Subroutines MEM and MEMW.
3.) Subroutine PRINT.
4.) Subroutine SCHEDULE.
5.) Start-up of simulation.
6.) Code implementing states and state transitions.

5.1.2.2.7.1 Declaration variables and tables.

These declarations are in lines 394-429. The 7-byte array V contains all the information to write one record of the binary file. In conjunction with the identically structured array OLDV it contains the information for one line of listing. The first 6 bytes of V (and of OLDV) are

structured like the binary file records. The supplemental 7-th byte keeps track of which Q-80 registers are uninitialized. Figure 5.1-16 shows the organization of V and OLDV. The last two bytes of V and OLDV can also be referenced by the name DEF and OLDDEF, respectively.

The following Q-80 registers are declared as 1-byte variables: B, C, D, E, H, L, A. The 2-byte variables BC, DE and HL are overlayed on the respective registers and thus facilitate register pair access. The Q-80 registers PC and SP are represented as 2-byte variables. The Q-80 flags S, Z, P, CY, OFL and INTE are represented by 1-byte variables. The Q-80 stimuli RESIN, HALT, HOLD and INT are represented as 1-byte variables (HALT is represented by the variable HLT).

A 3-byte array INTJAMWORD is provided to hold the 3-bytes to be jammed into the Q-80 in case of interrupt.

The variable OPCODE holds the first byte fetched during instruction fetch or the first byte jammed during interrupt jams. The 2-byte variable ADDR holds the 2nd and 3rd byte of instruction (fetched or jammed). The 2 bytes in ADDR are individually addressable by the variable names DATABYTE (least significant half) and ADDRH (most significant byte).

The 2-byte variable MDATA2 holds operands fetched from memory or popped from the stack. The 2 bytes are individually addressable by the variable names MDATA (least significant half) and MDATAH (most significant half).

The 2-byte variable SDATA2 holds results to be stored in memory or pushed onto the stack. The 2 bytes are individually addressable by the variable names SDATA (least significant half) and SDATAH (most significant half).

The 2-byte variable MADDR is overlayed on the first two bytes of array V. The 1-byte variable LASTMEMTOQ80 holds the last byte sent from the memory, from an I/O port, or jammed to the Q-80. The 2-byte variable INSTRADDR holds the location of the opcode of the currently executed instruction. (Note that PC is incremented 1, 2 or 3 times and thus no longer points to the opcode). The variable SEQUCOUNT is used in subroutine SCHEDULE and must be (relatively) global to that subroutine. Its use will be discussed in the description of subroutine SCHEDULE.

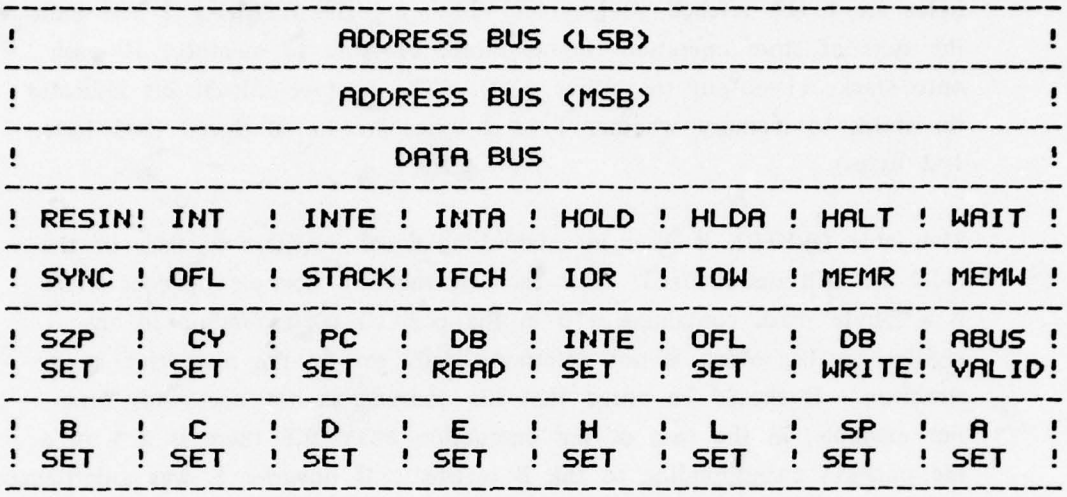


Figure 5.1-16 Organization of arrays V and OLDV.

The table TBL contains 256 bytes, one for each possible opcode. The 2 most significant bits specify whether it is a 1, 2 or 3-byte instruction (00=1 byte, 01=2-byte, 10=3-byte). The next 2 bits indicate the type of operand fetch (00=no operands to fetch, 01=fetch operand from memory, 10=pop operands from stack, 11=input operand from input port). The next bit indicates for operand fetches from memory, whether 1 or 2 bytes are to be fetched (0=1-byte, 1=2-byte). The following 2 bits indicate the type of store operation (00=no store, 01=store to memory, 10=push onto stack, 11=output to output port). The least significant bit indicates for stores to memory whether 1 or 2 bytes are to be stored (0=1 byte, 1=2 bytes).

The table DEFTBL is used in establishing which registers or flags of the Q-80 are still uninitialized. For each of the 256 possible opcodes there is a 2-byte mask containing a 0 in the position corresponding to any register or flag which is not deterministically set by the respective instruction. It should be noted that the converse is not necessarily true. For example, in the case of the instruction MOV B,E there is a 1 in the position corresponding to the B register. If however E was uninitialized, B will be uninitialized even after execution of this instruction (i.e. it will contain a random value).

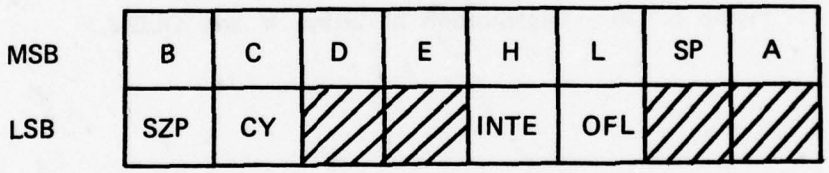


Figure 5.1-17 Bit assignment in an entry of table DEFTBL.

5.1.2.2.7.2 Procedures MEM and MEMW.

The function MEM and the subroutine MEMW are located in lines 430-451. MEM simulates memory-read accesses, while MEMW simulates memory write accesses.

The function MEM has one parameter ADDR. If ADDR points to a location past the end of simulated memory the simulation run is aborted after printing the message "***ATTEMPT TO ACCESS OUT OF BOUNDS MEMORY LOCATION". If the address is within the simulated address space the function returns the value of the respective simulated memory location.

Similarly, subroutine MEMW has 2 parameters ADDR and VAL. If ADDR is outside the simulated address space, the simulation run is aborted after printing the message "***ATTEMPT TO WRITE OUT OF BOUNDS MEMORY LOCATION". Otherwise the respective simulated memory location is set to the value VAL.

5.1.2.2.7.3 Subroutine PRINT.

This subroutine is located in lines 452-583. It prints one line containing the hexadecimal value of PC, the data bus coming into the Q-80, RESIN, INT, HOLD, HALT, the address bus, the data bus going from the Q-80, STACK, IFCH, IOR, IOW, MEMR, MEMW, INTE, INTA, HLDA, WAIT, the registers B, C, D, E, H, L, SP, A, the flags S, Z, P, CY, OFL. Wherever the corresponding bit in DEF or OLDDEF (which ever may be applicable) is 0 indicating a random value, blanks are sent to the printer.

5.1.2.2.7.4 Subroutine SCHEDULE.

This subroutine is located in lines 584-658. It sets the Q-80 stimuli RESIN, HALT, HOLD and INT according to the contents of the structure COMM based on the location of the opcode of the current instruction, the current state and the count that is kept in applicable cells of the structure COMM.

Execution of SCHEDULE begins with the copying of array V to OLDV. Next for all cells where COMM(i).LOC equals the location of the opcode of the current instruction (INSTRADDR), COMM(i).COUNT is incremented. The structure COMM is searched next until a cell COMM(i) is found where LOC=INSTRADDR and COUNT=TRIGGER.

If none is found (as is the case most of the time) RESIN, HALT, HOLD and INT are set to 0 before returning.

If the current state is IFCH1 or INTJAM1 then this is the beginning of a new instruction and SEQUCOUNT, the pointer to the 8 RHHI groups in SEQU is set to 0. Thereafter it is determined whether the current state dictates fetching RHHI from FIXED or from SEQU. RESIN, HLT, HOLD and INT are set accordingly before returning.

Note that SEQUCOUNT must be presrved until the next call of SCHEDULE. For that reason it had to be made a variable of larger scope than the procedure SCHEDULE.

5.1.2.2.7.5 Start-up of simulation.

In liner 659-661 the arrays V and OLDV are set to all 0's, except for the RESIN-bit in V. Subroutine PRINT is called to print the initial line. It is significant to note that all registers and flags are assumed to have random values. Execution proceeds into simulation of the RESET state.

5.1.2.2.7.6 Simulation of states and state transitions.

The blocks used to simulate each of the 24 states are quite similar. In each such block V is set to some value (depending on the state in question), DEF is set as a function of OLDDEF, V is output to the binary file by a call of ALPHA, the next set of stimuli is determined by a call to SCHEDULE, a line is printed on the listing by a call to PRINT, and the next state is decided. The subsequent state-by-state description covers only activities supplemental to the ones described above.

5.1.2.2.7.6.1 Simulation of state RESET.

The state RESET is simulated in lines 662-681. PC and INSTRADDR are set to 0 and the address bus and the data bus are floated.

5.1.2.2.7.6.2 Simulation of state IFCH1.

The state IFCH1 is simulated in lines 682-713. PC is placed on the address bus and both IFCH and MEMR are set to 1. INSTRADDR is set equal to PC, thus indicating the start of execution for a new instruction. The value read from simulated memory by the function MEM is assigned to OPCODE. Thereafter the program counter (PC) is incremented and OFL is set to 0. If OPCODE indicates that the instruction is one of the "return conditional" instructions, the decision of the next state is left to the pseudo-state COND.

5.1.2.2.7.6.3 Simulation of state IFCH2.

The state IFCH2 is simulated in lines 714-740. PC is placed on the address bus and both IFCH and MEMR are set to 1. The value read from simulated memory by a call of the function MEM is assigned to DATABYTE (which is the least significant half of the 2-byte variable ADDR). PC is incremented thereafter.

5.1.2.2.7.6.4 Simulation of the state IFCH3.

The state IFCH3 is simulated in lines 741-765. PC is placed on the address bus and both IFCH and MEMR are set to 1. The value read from simulated memory by a call to the function MEM is assigned to ADDRH (the most significant half of the 2-byte variable ADDR). Thereafter PC is incremented.

5.1.2.2.7.6.5 Simulation of the pseudo-state COND.

The pseudo-state COND is simulated in lines 766-781. Depending on the opcode and on the state of one of the Q-80 flags S, Z, P or CY (according to opcode) the next state is either POP1 or the pseudo-state EXEC.

The pseudo-state COND deals only with the execution of conditional return instructions. If the condition is met (e.g. Z=0 meets the condition for RNZ) the stack must be popped, therefore the next state is POP1. Otherwise the instruction is treated like a NOP and the next state is the pseudo-state EXEC.

5.1.2.2.7.6.6 Simulation of state OFCH1.

The state OFCH1 is simulated in lines 782-810. Depending on the opcode, one of the following is placed on the address bus:

- the register pair BC
- the register pair DE
- the register pair HL
- the 2-nd and 3-rd byte of the instruction.

MEMR is set to 1. The value read from simulated memory by a call of the function MEM is assigned to MDATA (the least significant half of the 2-byte variable MDATA2).

5.1.2.2.7.6.7 Simulation of state OFCH2.

The state OFCH2 is simulated in lines 811-829. The value placed on the address bus. MEMR is set to 1. The value read from simulated memory by a call of the function MEM is assigned to MDATAH (the most significant half of the 2-byte variable MDATA2).

5.1.2.2.7.6.8 Simulation of state POP1.

The state POP1 is simulated in lines 830-849. The value of SP is placed on the address bus. STACK and MEMR are set to 1. The value read from simulated memory by a call of the function MEM is assigned to MDATA (the least significant half of the 2-byte variable MDATA2) and the stack pointer (SP) is incremented.

5.1.2.2.7.6.9 Simulation of state POP2.

The state POP2 is simulated in lines 850-869. The value of SP is placed on the address bus.STACK and MEMR are set to 1. The value read from simulated memory by a call of the function MEM is assigned to MDATAH (the most significant half of the 2-byte variable MDATA2) and the stack pointer (SP) is incremented.

5.1.2.2.7.6.10 Simulation of state INPUT.

The state INPUT is simulated in lines 870-896. The second byte of the instruction (which was stored in the variable DATABYTE when state IFCH2 or INTJAM2 was simulated) is placed on the address bus padded with leading zeros. IOR is set to 1. The simulated input operation reads one byte from the Q-80 Input File into the variable MDATA (the least significant half of the 2-byte variable MDATA2). If the file does not exist (INSTATUS=13) or if the end-of-file has been reached (L=0) the message: "***ATTEMPT TO READ PAST END OF INPUT FILE." is printed.

5.1.2.2.7.6.11 Simulation of the pseudo-state EXEC.

The pseudo-state EXEC is simulated in lines 897-2203. It consists of a DO-CASE construct which implements the 256-way branch on OPCODE. Thus each possible opcode is processed separately. The DO-CASE construct is easy to follow and is thus virtually selfdocumenting. Operands fetched from memory in other states are in the 2-byte variable MDATA2 and the 2nd and 3rd byte of instructions are in the 2-byte variable ADDR. Results produced in EXEC which have to be written into the memory or output on a port are assigned to the 2-byte variable SDATA2.

HLT-instructions are simulated with irregular exits from the DO-CASE construct, branching to one of the states HALTI,HALTIE,IHHOLD,IEHHOLD or INTJAM1. All other instruction are simulated via a regular exit from the DO-CASE construct followed by branches to one of the states IFCH1, HALTE,HOLD,HHOLD, INTJAM1,STORE1,PUSH1 or OUTPUT.

5.1.2.2.7.6.12 Simulation of state STORE1.

The state STORE1 is simulated in lines 2304-2340. Depending on the opcode one of the following is placed on the address bus:

- the register pair BC
- the register pair DE
- the register pair HL
- the 2nd and 3rd byte of the instruction.

The value of SDATA (the least significant half of the 2-byte variable SDATA2) is placed on the data bus and MEMW is set to 1. The value of SDATA is written into simulated memory by a call of subroutine MEMW.

5.1.2.2.7.6.13 Simulation of state STORE2.

The state STORE2 is simulated in lines 2341-2367. The value placed on the address bus in state STORE1 is incremented and placed on the address bus. The value of the most significant half of the 2-byte variable SDATA2 is placed on the data bus and MEMW is set to 1. The value of the most significant half of SDATA2 is also written into simulated memory by a call of subroutine MEMW.

5.1.2.2.7.6.14 Simulation of state PUSH1.

The state PUSH1 is simulated in lines 2368-2387. The stack pointer (SP) is decremented first and its value is placed on the address bus. The most significant half of the 2-byte variable SDATA2 is placed on the data bus and both MEMW and STACK are set to 1. The most significant half of SDATA2 is also written into simulated memory by a call of subroutine MEMW.

5.1.2.2.7.6.15 Simulation of state PUSH2.

The state PUSH2 is simulated in lines 2388-2415. The stack pointer (SP) is decremented first and its value is placed on the address bus. The value of SDATA (the least significant half of the 2-byte variable SDATA2) is placed on the data bus and both MEMW and STACK are set to 1. The value of SDATA is also written into simulated memory by a call of subroutine MEMW.

5.1.2.2.7.6.16 Simulation of state OUTPUT.

The state OUTPUT is simulated in lines 2416-2441. The 2nd byte of the instruction, which represents the port number, is placed on the address bus, padded with leading zeros. The value of SDATA is placed o the data bus and IOW is set to 1.

5.1.2.2.7.6.17 Simulation of state HALTE.

The state HALTE is simulated in lines 2442-2465. The WAIT-bit is set to 1.

5.1.2.2.7.18 Simulation of state HOLD.

The state HOLD is simulated in lines 2466-2495. To simulate the floating of the address bus and of the data bus the corresponding simulated variables are set to all 1s. HLDA is set to 1. The value on the buses is stored in the temporary storage variables OLDADDR and LASTMEMTOQ80 and the buses are restored to their old value before the state transition.

5.1.2.2.7.6.19 Simulation of state HHOLD.

The state HHOLD is simulated in lines 2496-2525. To simulate the floating of the address bus and of the data bus the corresponding simulated variables are set to all 1s. HLDA and WAIT are set to 1. The value on the buses is stored in the temporary storage variables OLDADDR and LASTMEMTOQ80 and the buses are restored to their old value before the state transition.

5.1.2.2.7.6.20 Simulation of the state HALTI.

The state HALTI is simulated in lines 2526-2554. The WAIT-bit is set to 1. If the call of subroutine SCHEDULE results in RESIN=0, HALT=0, HOLD=0 and INT.INTE=0 then the simulation run comes to a normal end. The paper in the printer is advanced two pages by two calls of subroutine PAGE and the program is terminated by a call of EXIT.

5.1.2.2.7.6.21 Simulation of state HALTE.

The state HALTIE is simulated in lines 2555-2583. The WAIT-bit is set to 1. If the call of subroutine SCHEDULE results in RESIN=0, HALT=0, HOLD=0 and INTE.INTE=0 their the simulation run comes to a normal end. The paper in the printer is advanced two pages by two calls of subroutine PAGE and the program is terminated by a call of EXIT.

5.1.2.2.7.6.22 Simulation of state IHHOLD.

The state IHHOLD is simulated in lines 2584-2618. Both WAIT and HLDA are set to 1. To simulate the floating of the address bus and of the data bus the corresponding simulation variables are set to all 1's. The value on the buses is stored in the temporary storage variables OLDADDR and LASTMEMTOQ80 and the buses are restored to their old values before the state transition.

5.1.2.2.7.6.23 Simulation of state IEHHOLD.

The state IEHHOLD is simulated in lines 2619-2650. Both WAIT and HLDA are set to 1. To simulate the floating of the address bus and of the data bus the corresponding simulation variables are set to all 1s. The value on the buses is saved in the temporary storage variables OLDADDR and LASTMEMTOQ80 and the buses are restored to their old values before state transition.

5.1.2.2.7.6.24 Simulation of state INTJAM1.

The state INTJAM1 is simulated in lines 2651-2671. First INTE is set to 0. PC is placed on the address bus and both IFCH and INTA are set to 1. The jamming of the 1st byte is simulated by assigning INTJAMWORD(0) to OPCODE. OFL is set to 0.

5.1.2.2.7.6.25 Simulation of state INTJAM2.

The state INTJAM2 is simulated in lines 2672-2683. Again, PC is placed on the address bus, without having been incremented, and both IFCH and INTA are set to 1. The jamming of the 2nd byte is simulated by assigning INTJAMWORD(1) to DATABYTE (the least significant half of the 2-byte variable ADDR).

5.1.2.2.7.6.26 Simulation of state INTJAM3.

The state INTJAM3 is simulated in lines 2684-2698. Again, PC is placed on the address bus, without having been incremented, and both IFCH and

INTA are set to 1. The jamming of the 3rd byte is simulated by assigning INTJAMWORD(2) to ADDRH (the most significant half of the 2-byte variable ADDR).

5.1.2.2.7.6.27 Simulation of state transitions.

State transitions are simulated by a series of conditionally executed GOTO-instructions located at the end of each of the 26 blocks described so far in this section (5.1.2.2.7.6.1–5.1.2.2.7.6.26). State transitions are described in the Q-80 Simulator – User's Reference (5.1.2.1).

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE Q80SIM
OBJECT MODULE PLACED IN :F1:Q80SIM.OBJ
COMPILER INVOKED BY:  PLM80 :F1:Q80SIM.PLM


                $PRINT(:LP:) PAGEWIDTH(80) TITLE('Q-80 SIMULATOR')
 1              Q80SIM: DO;
                $INCLUDE(INCLUD.DCL)
 2   1  =       OPEN:   PROCEDURE(AFTPTR,FNPTR,ACCCODE,ECHOAFT,STATUS) EXTERNAL;
 3   2  =                 DECLARE(AFTPTR,FNPTR,ACCCODE,ECHOAFT,STATUS) ADDRESS;
 4   2  =               END OPEN;
 5   1  =       CLOSE:  PROCEDURE(AFT,STATUS) EXTERNAL;
 6   2  =                 DECLARE(AFT,STATUS) ADDRESS;
 7   2  =               END CLOSE;
 8   1  =       READ:   PROCEDURE(AFT,BUFPTR,LENGTH,COUNTPTR,STATUS) EXTERNAL;
 9   2  =                 DECLARE(AFT,BUFPTR,LENGTH,COUNTPTR,STATUS) ADDRESS;
10   2  =               END READ;
11   1  =       WRITE:  PROCEDURE(AFT,BUFPTR,LENGTH,STATUS) EXTERNAL;
12   2  =                 DECLARE(AFT,BUFPTR,LENGTH,STATUS) ADDRESS;
13   2  =               END WRITE;
14   1  =       SEEK:   PROCEDURE(AFT,OPER,BLOCKNOPTR,BYTENOPTR,STATUS) EXTERNAL;
15   2  =                 DECLARE(AFT,OPER,BLOCKNOPTR,BYTENOPTR,STATUS) ADDRESS;
16   2  =               END SEEK;
17   1  =       DELETE: PROCEDURE(AFT,STATUS) EXTERNAL;
18   2  =                   DECLARE(AFT,STATUS) ADDRESS;
19   2  =                 END DELETE;
20   1  =       ERROR:  PROCEDURE(ERRNO,STATUS) EXTERNAL;
21   2  =                 DECLARE(ERRNO,STATUS) ADDRESS;
22   2  =               END ERROR;
23   1  =       EXIT:   PROCEDURE EXTERNAL;
24   2  =               END EXIT;
     =
25   1  =       ALPHA:  PROCEDURE(AFT,LENGTH,BUFPTR) EXTERNAL;
26   2  =                 DECLARE(AFT,LENGTH,BUFPTR) ADDRESS;
27   2  =               END ALPHA;
28   1  =       HEX:    PROCEDURE(AFT,LENGTH,BUFPTR) EXTERNAL;
29   2  =                 DECLARE(AFT,LENGTH,BUFPTR) ADDRESS;
30   2  =               END HEX;
31   1  =       INTEG:  PROCEDURE(AFT,BITE) EXTERNAL;
32   2  =                 DECLARE AFT ADDRESS, BITE BYTE;
33   2  =               END INTEG;
34   1  =       INTEGD: PROCEDURE(AFT,TWOBITES) EXTERNAL;
35   2  =                 DECLARE(AFT,TWOBITES) ADDRESS;
36   2  =               END INTEGD;
37   1  =       PAGE:   PROCEDURE(AFT) EXTERNAL;
38   2  =                 DECLARE AFT ADDRESS;
39   2  =               END PAGE;
40   1  =       PPAGE:  PROCEDURE(AFT) EXTERNAL;
41   2  =                 DECLARE AFT ADDRESS;
42   2  =               END PPAGE;
43   1  =       SKIP:   PROCEDURE(AFT) EXTERNAL;
44   2  =                 DECLARE(AFT) ADDRESS;
45   2  =               END SKIP;
46   1  =       SPACE:  PROCEDURE(AFT,LENGTH) EXTERNAL;
47   2  =                 DECLARE(AFT,LENGTH) ADDRESS;
48   2  =               END SPACE;
```

```
      =
49    1 =   EQUAL:  PROCEDURE(P1,P2,L) BYTE EXTERNAL;
50    2 =             DECLARE(P1,P2) ADDRESS, L BYTE;
51    2 =           END EQUAL;
52    1 =   SETBIT: PROCEDURE(P,I,L,V) EXTERNAL;
53    2 =             DECLARE P ADDRESS, (I,L,V) BYTE;
54    2 =           END SETBIT;
55    1 =   EQUBIT: PROCEDURE(P,I,L,V) BYTE EXTERNAL;
56    2 =             DECLARE P ADDRESS, (I,L,V) BYTE;
57    2 =           END EQUBIT;
58    1 =   SUBSTR: PROCEDURE(P,I,L) BYTE EXTERNAL;
59    2 =             DECLARE P ADDRESS, (I,L) BYTE;
60    2 =           END SUBSTR;
```

```
            $EJECT
                  /*****************************************************************/
                  /*   DECLARATION OF GLOBAL VARIABLES AND PROCEDURES.             */
                  /*****************************************************************/

 61   1           DECLARE (BI, BO, LP, Q80IN, SCH) ADDRESS,
                       KB  ADDRESS DATA(1),
                       CRT ADDRESS DATA(0);
 62   1           DECLARE CR  BYTE DATA(0DH);
 63   1           DECLARE TAB BYTE DATA(09H);
 64   1           DECLARE PAGENO ADDRESS;
 65   1           DECLARE LINENO BYTE;
 66   1           DECLARE FN(15) BYTE;
 67   1           DECLARE COMM(50) STRUCTURE(LOC      ADDRESS,
                                             TRIGGER  ADDRESS,
                                             COUNT    ADDRESS,
                                             FIXED(8) BYTE,
                                             SEQU(4)  BYTE,
                                             JAM(3)   BYTE);
 68   1           DECLARE INSTATUS ADDRESS;
 69   1           HEADER:PROCEDURE(SW);
 70   2                 DECLARE SW BYTE;
 71   2                 DECLARE (DATEAFT, L, S) ADDRESS;
 72   2                 DECLARE DATE(11) BYTE;
 73   2                 IF SW THEN CALL PPAGE(LP);
 75   2                       ELSE CALL PAGE(LP);
 76   2                 CALL ALPHA(LP, 20, .(0EH, '            QUESTRON'));
 77   2                 CALL SKIP(LP);
 78   2                 CALL SKIP(LP);
 79   2                 CALL ALPHA(LP, 24, .(0EH, '         Q-80 SIMULATOR'));
 80   2                 CALL SKIP(LP);
 81   2                 CALL SPACE(LP, 70);
 82   2                 CALL ALPHA(LP, 4, .('PAGE'));
 83   2                 CALL INTEGD(LP, PAGENO);
 84   2                 CALL SKIP(LP);
 85   2                 CALL ALPHA(LP, 41, .(0EH, '--------------------',
                                                 '--------------------'));
 86   2                 CALL SKIP(LP);
 87   2                 CALL SKIP(LP);
 88   2                 IF SW
                           THEN DO;
 90   3                       CALL SPACE(LP, 8);
 91   3                       CALL ALPHA(LP, 14, .FN);
 92   3                       CALL SPACE(LP, 20);
 93   3                       CALL OPEN(.DATEAFT, .('DATE '), 1, 0, .S);
 94   3                       CALL READ(DATEAFT, .DATE, 11, .L, .S);
 95   3                       CALL CLOSE(DATEAFT, .S);
 96   3                       CALL ALPHA(LP, 11, .DATE);
 97   3                       CALL SKIP(LP);
 98   3                       CALL SKIP(LP);
 99   3                       CALL ALPHA(LP, 41, .(0EH, '---------------',
                                                     '-------------------------'));
100   3                       CALL SKIP(LP);
101   3                       CALL SKIP(LP);
102   3                       GOTO ENDHEADER;
103   3                       END;
104   2                 CALL ALPHA(LP, 78, .('-------------------------',
```

```
                                        '-------------------------',
                                        '-------------------------'));
 105   2                  CALL SKIP(LP);
 106   2                  CALL ALPHA(LP,78,.('!    !   R I H H!        S',
                                        ' I          I I H W!         ',
                                        '                 !         !'));
 107   2                  CALL SKIP(LP);
 108   2                  CALL ALPHA(LP,78,.('!    !   E N O A!        T',
                                        ' F          N N L A!        R',
                                        'EGISTERS        !  FLAGS  !'));
 109   2                  CALL SKIP(LP);
 110   2                  CALL ALPHA(LP,78,.('!    !   S T L L!        A',
                                        ' C I/0 MEM T T D I!         ',
                                        '                 !        O!'));
 111   2                  CALL SKIP(LP);
 112   2                  CALL ALPHA(LP,78,.('!    !   I   D T!        C',
                                        ' H            E A A T!         ',
                                        '                 !      C F!'));
 113   2                  CALL SKIP(LP);
 114   2                  CALL ALPHA(LP,78,.('! PC !DB N       !ADDR DB K',
                                        '   R W R W          ! BC   D',
                                        'E   HL   SP  A !S Z P Y L!'));
 115   2                  CALL SKIP(LP);
 116   2                  CALL ALPHA(LP,78,.('!----!----------!---------',
                                        '------------------!-------',
                                        '----------------!---------!'));
 117   2                  CALL SKIP(LP);
 118   2     ENDHEADER:IF SW THEN LINENO = 9;
 120   2                    ELSE LINENO = 12;
 121   2                  PAGENO = PAGENO + 1;
 122   2                  END HEADER;
 123   1        EOL:      PROCEDURE;
 124   2                  LINENO = LINENO + 1;
 125   2                  CALL SKIP(LP);
 126   2                  IF LINENO <= 61 THEN RETURN;
 128   2                                  ELSE CALL HEADER(0);
 129   2                  END EOL;
```

```
                $EJECT
                        /*****************************/
                        /*  PROLOGUE ACTIVITIES  */
                        /*****************************/

130   1                   DO;
131   2                   DECLARE LINE(256) BYTE;
132   2                   DECLARE (LENGTH, STATUS) ADDRESS;
133   2                   DECLARE (LINEPTR, FNPTR, I) BYTE;

                        /* READ REMAINDER OF COMMAND LINE */

134   2                   CALL READ(KB,.LINE,255,.LENGTH,.STATUS);
135   2                   IF LENGTH=2 THEN CALL EXIT;

                        /* IGNORE LEADING BLANKS */

137   2                   LINEPTR = 0;
138   2                   DO WHILE LINE(LINEPTR)=' ';
139   3                   LINEPTR = LINEPTR + 1;
140   3                   END;

                        /* COPY FILENAME FROM LINE TO FN */

141   2                   FNPTR = 0;
142   2                   DO WHILE LINE(LINEPTR)<>' ' AND LINE(LINEPTR)<>0DH
                                                    AND  LINEPTR<14;
143   3                   FN(FNPTR) = LINE(LINEPTR);
144   3                   LINEPTR = LINEPTR + 1;
145   3                   FNPTR   = FNPTR   + 1;
146   3                   END;

                        /* FILL IN REMAINDER OF FN WITH BLANKS */

147   2                   DO I = FNPTR TO 14;
148   3                   FN(I) = ' ';
149   3                   END;

                        /* OPEN SOURCE FILE AND CHECK FOR ERRORS */

150   2                   CALL OPEN(.BI,.FN,1,0,.STATUS);
151   2                   IF STATUS <> 0
                             THEN DO;
153   3                          CALL ALPHA(CRT,32,.('*** CAN NOT OPEN FILE.  ',
                                                    'STATUS = '));
154   3                          CALL HEX  (CRT,2,.STATUS+1);
155   3                          CALL HEX  (CRT,2,.STATUS);
156   3                          CALL SKIP (CRT);
157   3                          CALL EXIT;
158   3                          END;

                        /* OPEN :LP: FILE AND PRINT FIRST HEADER */

159   2                   CALL OPEN(.LP,.(':LP: '),2,0,.STATUS);
160   2                   PAGENO = 1;
161   2                   CALL HEADER(1);
```

```
                    /* OPEN OUTPUT FILE AND Q80-INPUT FILE */

162   2               I = 1;
163   2               DO WHILE (I<=10) AND (FN(I) <> '.') AND (FN(I) <> ' ');
164   3               I = I + 1;
165   3               END;
166   2               CALL MOVE(4,.('.BIN'),.FN+I);
167   2               CALL OPEN(.BO,.FN,2,0,.STATUS);
168   2               CALL MOVE(4,.('.IN '),.FN+I);
169   2               CALL OPEN(.Q80IN,.FN,1,0,.INSTATUS);

170   2               END;
```

```
                $EJECT
                        /************************/
                        /*  READ OBJECT FILE  */
                        /************************/

171   1                 DO;
172   2                 DECLARE (L,S) ADDRESS;
173   2                 DECLARE BUFF(256) BYTE;
174   2                 DECLARE TYPE BYTE;
175   2                 DECLARE LENGTH ADDRESS;
176   2                 DECLARE P ADDRESS;

177   2                 P = 0;
178   2                 CALL READ(BI,.TYPE,1,.L,.S);
179   2                 DO WHILE L>0;
180   3                 CALL READ(BI,.LENGTH,2,.L,.S);
181   3                 IF TYPE = 6
                          THEN DO;
183   4                       CALL READ(BI,.BUFF,1,.L,.S);
184   4                       CALL READ(BI,.P,2,.L,.S);
185   4                       IF .MEMORY + P + LENGTH - 4 > 0F6B0H
                                THEN DO;
187   5                             CALL ALPHA(LP,50,.('*** ATTEMPT TO LOAD',
                                         ' OUT OF BOUNDS MEMORY LOCATION.'));
188   5                             CALL EOL;
189   5                             CALL EXIT;
190   5                             END;
191   4                       CALL READ(BI,.MEMORY+P,LENGTH-3,.L,.S);
192   4                       P = P + LENGTH - 5;
193   4                       END;
194   3                   ELSE DO;
195   4                       CALL READ(BI,.BUFF,LENGTH,.L,.S);
196   4                       END;
197   3                 CALL READ(BI,.TYPE,1,.L,.S);
198   3                 END;
199   2                 CALL CLOSE(BI,.S);
200   2                 END;
```

```
$EJECT

                        /**********************/
                        /*  READ SCHEDULE  */
                        /**********************/

201   1             DO;
202   2             DECLARE BB ADDRESS;
203   2             DECLARE LINE(80) BYTE;
204   2             DECLARE PRLINE(81) BYTE;
205   2             DECLARE (I,J,K,M,N) BYTE;
206   2             DECLARE RHHI BYTE;
207   2             DECLARE (L,S) ADDRESS;
208   2             HEX:    PROCEDURE(A) ADDRESS;
209   3                     DECLARE A BYTE;
210   3                     IF ((A < 'A') OR (A > 'F')) AND
                             ((A < '0') OR (A > '9'))
                             THEN DO;
212   4                           CALL ALPHA(CRT,17,.('*** NON-HEX DIGIT'));
213   4                           CALL SKIP (CRT);
214   4                           CALL EXIT;
215   4                           END;
216   3                     IF (A AND 0F0H) = 40H THEN RETURN (A AND 0FH) + 9;
218   3                                           ELSE RETURN (A AND 0FH);
219   3                     END HEX;

                    /* OPEN SCHEDULE FILE */

220   2                 I = 1;
221   2                 DO WHILE (I<=10) AND (FN(I) <> '.') AND (FN(I) <> ' ');
222   3                 I = I + 1;
223   3                 END;
224   2                 CALL MOVE(4,.('.SCH'),.FN+I);
225   2                 CALL OPEN(.BB,.(':BB: '),2,0,.S);
226   2                 CALL OPEN(.SCH,.FN,1,BB,.S);

                    /* SET ALL COMM-TABLE ENTRIES OFF */

227   2                 DO I = 0 TO 49;
228   3                 CALL MOVE(21,.(0FFH,0FFH,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                          0,0,0),.COMM+I*21);
229   3                 END;

                    /* THE FOLLOWING LOOP PROCESSES ONE LINE OF THE SCHEDULE */
                    /*                FILE IN EACH ITERATION.                */

230   2                 CALL READ(SCH,.LINE,80,.L,.S);
231   2                 I = 0;

232   2                 DO WHILE (L > 0) AND (S <> 13) AND (I < 50);
233   3                 CALL MOVE(80,.('                                        ',
                                     '                                        ',
                                     '                          '),.PRLINE);
234   3                 K = 0;
235   3                 DO J = 0 TO L-3;
236   4                 IF LINE(J) = TAB THEN K = K + 8 - (K MOD 8);
238   4                                  ELSE DO;
```

```
239   5                                 PRLINE(K) = LINE(J);
240   5                                 K = K + 1;
241   5                                 END;
242   4             IF K > 80 THEN K = 80;
244   4             END;
245   3             IF K > 79 THEN K = 80;
247   3                       ELSE K = K + 1;
248   3             CALL ALPHA(CRT,K,.PRLINE);
249   3             CALL SKIP (CRT);
250   3             CALL ALPHA(LP,K,.PRLINE);
251   3             CALL EOL;
252   3             COMM(I).LOC = SHL(HEX(LINE(0)),12) OR
                                  SHL(HEX(LINE(1)), 8) OR
                                  SHL(HEX(LINE(2)), 4) OR
                                      HEX(LINE(3));
253   3             IF LINE(4) <> TAB
                      THEN DO;
255   4                       CALL ALPHA(CRT,15,.('*** MISSING TAB'));
256   4                       CALL SKIP (CRT);
257   4                       CALL EXIT;
258   4                       END;
259   3             J = 5;          /* POINT TO BEGINNING OF TRIGGER FIELD */
260   3                       DO WHILE (LINE(J) <> TAB) AND (J < 10);
261   4                       IF (LINE(J) < '0') OR (LINE(J) > '9')
                                 THEN DO;
263   5                                 CALL ALPHA(CRT,17,.('*** ILLEGAL DIGIT'));
264   5                                 CALL SKIP (CRT);
265   5                                 CALL EXIT;
266   5                                 END;
267   4                             ELSE COMM(I).TRIGGER = COMM(I).TRIGGER * 10 +
                                                         (LINE(J) AND 0FH);
268   4                       J = J + 1;
269   4                       END;
270   3             IF LINE(J) <> TAB
                      THEN DO;
272   4                       CALL ALPHA(CRT,15,.('*** MISSING TAB'));
273   4                       CALL SKIP (CRT);
274   4                       CALL EXIT;
275   4                       END;
276   3             N = 0;
277   3             DO WHILE (N < I) AND NOT EQUAL(.COMM+I*21,.COMM+N*21,4);
278   4             N = N + 1;
279   4             END;  /* N = I  IF NO MATCHING ENTRY FOUND */
                        /* OTHERWISE N POINTS TO MATCHING ENTRY */
280   3             IF LINE(J+5) = '@'
                      THEN DO;
282   4                       RHHI = 0;
283   4                       DO K = J+1 TO J+4;
284   5                       IF (LINE(K) <> '0') AND (LINE(K) <> '1')
                                 THEN DO;
286   6                                 CALL ALPHA(CRT,24,.('*** ILLEGAL BINARY',
                                                         ' DIGIT'));
287   6                                 CALL SKIP (CRT);
288   6                                 CALL EXIT;
289   6                                 END;
290   5                       RHHI = SHL(RHHI,1) OR (LINE(K) AND 01H);
291   5                       END;
```

```
292  4                    J = J + 6;  /* POINTS TO BEGINNING OF STATE */
293  4                    IF EQUAL(.('IFCH1'),.LINE+J,5)
                             THEN CALL SETBIT(.COMM+N*21+6,0,4,RHHI);
295  4                    IF EQUAL(.('IFCH2'),.LINE+J,5)
                             THEN CALL SETBIT(.COMM+N*21+6,4,4,RHHI);
297  4                    IF EQUAL(.('IFCH3'),.LINE+J,5)
                             THEN CALL SETBIT(.COMM+N*21+6,8,4,RHHI);
299  4                    IF EQUAL(.('OFCH1'),.LINE+J,5)
                             THEN CALL SETBIT(.COMM+N*21+6,12,4,RHHI);
301  4                    IF EQUAL(.('OFCH2'),.LINE+J,5)
                             THEN CALL SETBIT(.COMM+N*21+6,16,4,RHHI);
303  4                    IF EQUAL(.('POP1'),.LINE+J,4)
                             THEN CALL SETBIT(.COMM+N*21+6,20,4,RHHI);
305  4                    IF EQUAL(.('POP2'),.LINE+J,4)
                             THEN CALL SETBIT(.COMM+N*21+6,24,4,RHHI);
307  4                    IF EQUAL(.('INPUT'),.LINE+J,5)
                             THEN CALL SETBIT(.COMM+N*21+6,28,4,RHHI);
309  4                    IF EQUAL(.('STORE1'),.LINE+J,6)
                             THEN CALL SETBIT(.COMM+N*21+6,32,4,RHHI);
311  4                    IF EQUAL(.('STORE2'),.LINE+J,6)
                             THEN CALL SETBIT(.COMM+N*21+6,36,4,RHHI);
313  4                    IF EQUAL(.('PUSH1'),.LINE+J,5)
                             THEN CALL SETBIT(.COMM+N*21+6,40,4,RHHI);
315  4                    IF EQUAL(.('PUSH2'),.LINE+J,5)
                             THEN CALL SETBIT(.COMM+N*21+6,44,4,RHHI);
317  4                    IF EQUAL(.('OUTPUT'),.LINE+J,6)
                             THEN CALL SETBIT(.COMM+N*21+6,48,4,RHHI);
319  4                    IF EQUAL(.('INTJAM1'),.LINE+J,7)
                             THEN CALL SETBIT(.COMM+N*21+6,52,4,RHHI);
321  4                    IF EQUAL(.('INTJAM2'),.LINE+J,7)
                             THEN CALL SETBIT(.COMM+N*21+6,56,4,RHHI);
323  4                    IF EQUAL(.('INTJAM3'),.LINE+J,7)
                             THEN CALL SETBIT(.COMM+N*21+6,60,4,RHHI);
325  4                    IF NOT EQUAL(.('IFCH1'  ),.LINE+J,5) AND
                             NOT EQUAL(.('IFCH2'  ),.LINE+J,5) AND
                             NOT EQUAL(.('IFCH3'  ),.LINE+J,5) AND
                             NOT EQUAL(.('OFCH1'  ),.LINE+J,5) AND
                             NOT EQUAL(.('OFCH2'  ),.LINE+J,5) AND
                             NOT EQUAL(.('POP1'   ),.LINE+J,4) AND
                             NOT EQUAL(.('POP2'   ),.LINE+J,4) AND
                             NOT EQUAL(.('INPUT'  ),.LINE+J,5) AND
                             NOT EQUAL(.('STORE1' ),.LINE+J,6) AND
                             NOT EQUAL(.('STORE2' ),.LINE+J,6) AND
                             NOT EQUAL(.('PUSH1'  ),.LINE+J,5) AND
                             NOT EQUAL(.('PUSH2'  ),.LINE+J,5) AND
                             NOT EQUAL(.('OUTPUT' ),.LINE+J,6) AND
                             NOT EQUAL(.('INTJAM1'),.LINE+J,7) AND
                             NOT EQUAL(.('INTJAM2'),.LINE+J,7) AND
                             NOT EQUAL(.('INTJAM3'),.LINE+J,7)
                             THEN DO;
327  5                            CALL ALPHA(CRT,17,.('*** ILLEGAL STATE'));
328  5                            CALL SKIP (CRT);
329  5                            CALL EXIT;
330  5                            END;
331  4                    DO WHILE (LINE(J) <> TAB) AND (LINE(J) <> CR);
332  5                    J = J + 1;
333  5                    END;
```

```
334   4                    END;
335   3               ELSE DO;
336   4                    IF NOT EQUAL(.COMM+N*21+14,.(0,0,0,0),4)
                              THEN DO;
338   5                         CALL ALPHA(CRT,20,.('*** RESPECIFIED ',
                                                     'SEQU'));
339   5                         CALL SKIP (CRT);
340   5                         CALL EXIT;
341   5                         END;
342   4                    LINE(J) = ',';      /* TO MAKE LOOP EASIER */
343   4                    M = 0;
344   4                    DO WHILE (LINE(J) = ',') AND (M < 8);
345   5                    RHHI = 0;
346   5                         DO K = J+1 TO J+4;
347   6                         IF (LINE(K) <> '0') AND (LINE(K) <> '1')
                                    THEN DO;
349   7                              CALL ALPHA(CRT,24,.('*** ILLEGAL',
                                                  ' BINARY DIGIT'));
350   7                              CALL SKIP (CRT);
351   7                              CALL EXIT;
352   7                              END;
353   6                         RHHI = SHL(RHHI,1) OR (LINE(K) AND 01H);
354   6                         END;
355   5                    CALL SETBIT(.COMM+N*21+14,M*4,4,RHHI);
356   5                    M = M + 1;
357   5                    J = J + 5;
358   5                    END;
359   4                    IF (LINE(J) <> TAB) AND (LINE(J) <> CR)
                              THEN DO;
361   5                         CALL ALPHA(CRT,53,.('*** SEQU NOT PROP',
                                       'ERLY TERMINATED.  TAB OR CR EX',
                                       'PECTED.'));
362   5                         CALL SKIP (CRT);
363   5                         CALL EXIT;
364   5                         END;
365   4                    END;

366   3               DO WHILE LINE(J) = TAB;
367   4               J = J + 1;
368   4               END;
369   3               IF NOT EQUAL(.('JAM='),.LINE+J,4) AND
                           NOT EQUAL(.CR      ,.LINE+J,1)
                           THEN DO;
371   4                    CALL ALPHA(CRT,32,.('*** UNRECOGNIZED ',
                                                'TEXT AFTER SEQU'));
372   4                    CALL SKIP (CRT);
373   4                    CALL EXIT;
374   4                    END;
375   3               IF EQUAL(.('JAM='),.LINE+J,4)
                           THEN DO;
377   4                    J = J + 4;
378   4                    CALL SETBIT(.COMM+N*21+18, 0,4,
                                         HEX(LINE(J  )));
379   4                    CALL SETBIT(.COMM+N*21+18, 4,4,
                                         HEX(LINE(J+1)));
380   4                    CALL SETBIT(.COMM+N*21+18, 8,4,
                                         HEX(LINE(J+2)));
```

```
381   4                    CALL SETBIT(.COMM+N*21+18,12,4,
                                        HEX(LINE(J+3)));
382   4                    CALL SETBIT(.COMM+N*21+18,16,4,
                                        HEX(LINE(J+4)));
383   4                    CALL SETBIT(.COMM+N*21+18,20,4,
                                        HEX(LINE(J+5)));
384   4                    END;
385   3                 IF N = I
                           THEN I = I + 1;
387   3                    ELSE CALL MOVE(4,.(0FFH,0FFH,0,0),.COMM+I*21);
388   3                 CALL READ(SCH,.LINE,80,.L,.S);
389   3                 END;

                      /* CLOSE SCHEDULE FILE */

390   2                 CALL HEADER(0);
391   2                 CALL CLOSE(SCH,.S);
392   2                 CALL CLOSE(BB,.S);
393   2                 END;
```

```
$EJECT

                /*************************/
                /*   Q-80 SIMULATION  */
                /*************************/

394   1         DO;
395   2         DECLARE V(7) BYTE;
396   2         DECLARE DEF ADDRESS AT(.V+5);

          /*  ---------------------------------------------------------
              !                  ADDRESS BUS (LSB)                    !
              ---------------------------------------------------------
              !                  ADDRESS BUS (MSB)                    !
              ---------------------------------------------------------
              !                     DATA BUS                          !
              ---------------------------------------------------------
              ! RESIN! INT  ! INTE ! INTA ! HOLD ! HLDA ! HALT ! WAIT !
              ---------------------------------------------------------
              ! SYNC ! OFL  ! STACK! IFCH ! IOR  ! IOW  ! MEMR ! MEMW !
              ---------------------------------------------------------
              ! SZP  !  CY  !  PC  !  DB  ! INTE ! OFL  !  DB  ! ABUS !
              ! SET  ! SET  ! SET  ! READ ! SET  ! SET  ! WRITE! VALID!
              ---------------------------------------------------------
              !  B   !  C   !  D   !  E   !  H   !  L   !  SP  !  A   !
              ! SET  ! SET  ! SET  ! SET  ! SET  ! SET  ! SET  ! SET  !
              --------------------------------------------------------- */

397   2         DECLARE OLDV(7) BYTE;
398   2         DECLARE OLDDEF ADDRESS AT(.OLDV+5);

399   2         DECLARE BC ADDRESS;
400   2         DECLARE B  BYTE AT(.BC+1);
401   2         DECLARE C  BYTE AT(.BC);
402   2         DECLARE DE ADDRESS;
403   2         DECLARE D  BYTE AT(.DE+1);
404   2         DECLARE E  BYTE AT(.DE);
405   2         DECLARE HL ADDRESS;
406   2         DECLARE H  BYTE AT(.HL+1);
407   2         DECLARE L  BYTE AT(.HL);
408   2         DECLARE A  BYTE;
409   2         DECLARE SP ADDRESS;
410   2         DECLARE PC ADDRESS;

411   2         DECLARE (S,Z,P,CY,OFL,INTE) BYTE;

412   2         DECLARE (RESIN,INT,HOLD,HLT) BYTE;
413   2         DECLARE INTJAMWORD(3) BYTE;

414   2         DECLARE OPCODE BYTE;
415   2         DECLARE ADDR ADDRESS;
416   2         DECLARE DATABYTE BYTE AT(.ADDR);
417   2         DECLARE ADDRH    BYTE AT(.ADDR+1);
418   2         DECLARE MDATA2 ADDRESS;
419   2         DECLARE MDATA  BYTE AT(.MDATA2);
420   2         DECLARE MDATAH BYTE AT(.MDATA2+1);
421   2         DECLARE SDATA2 ADDRESS;
```

```
422   2           DECLARE SDATA  BYTE AT(.SDATA2);
423   2           DECLARE SDATAH BYTE AT(.SDATA+1);

424   2           DECLARE MADDR ADDRESS AT(.V);
425   2           DECLARE LASTMEMTOQ80 BYTE;
426   2           DECLARE INSTRADDR ADDRESS;
427   2           DECLARE SEQUCOUNT BYTE;
```

```
            $EJECT

428   2             DECLARE TBL(256) BYTE DATA(
                /*00*/  00H, 80H, 02H, 00H, 00H, 00H, 40H, 00H,
                        00H, 00H, 10H, 00H, 00H, 00H, 40H, 00H,
                /*10*/  00H, 80H, 02H, 00H, 00H, 00H, 40H, 00H,
                        00H, 00H, 10H, 00H, 00H, 00H, 40H, 60H,
                /*20*/  00H, 80H, 83H, 00H, 00H, 00H, 40H, 00H,
                        00H, 00H, 98H, 00H, 00H, 00H, 40H, 00H,
                /*30*/  00H, 80H, 82H, 00H, 12H, 12H, 42H, 00H,
                        00H, 00H, 90H, 00H, 00H, 00H, 40H, 00H,
                /*40*/  00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                        00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                /*50*/  00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                        00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                /*60*/  00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                        00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                /*70*/  02H, 02H, 02H, 02H, 02H, 02H, 00H, 02H,
                        00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                /*80*/  00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                        00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                /*90*/  00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                        00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                /*A0*/  00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                        00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                /*B0*/  00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                        00H, 00H, 00H, 00H, 00H, 00H, 10H, 00H,
                /*C0*/  00H, 20H, 80H, 80H, 84H, 04H, 40H, 04H,
                        00H, 20H, 80H, 00H, 84H, 84H, 40H, 04H,
                /*D0*/  00H, 20H, 80H, 40H, 84H, 04H, 40H, 04H,
                        00H, 00H, 80H, 40H, 84H, 00H, 40H, 04H,
                /*E0*/  00H, 20H, 80H, 24H, 84H, 04H, 40H, 04H,
                        00H, 00H, 80H, 00H, 84H, 00H, 40H, 04H,
                /*F0*/  00H, 20H, 80H, 00H, 84H, 04H, 40H, 04H,
                        00H, 00H, 80H, 00H, 84H, 00H, 40H, 04H);

                /*      ---------------------------------------------
                        ! IF1 ! IF0 ! OF1 ! OF0 ! DFC ! ST1 ! ST0 ! DST !
                        ---------------------------------------------

                        IF1 = 3-BYTE INSTRUCTION
                        IF0 = 2-BYTE INSTRUCTION

                        OF=00       NO OPERANDS TO FETCH
                        OF=01       OPERAND FETCHED FROM MEMORY (NOT STACK)
                        OF=10       OPERAND POPPED FROM STACK
                        OF=11       OPERAND INPUT

                        DFC = 2-BYTE OPERAND FETCHED FROM MEMORY (NOT STACK)

                        ST=00       NO RESULT TO STORE
                        ST=01       RESULT STORED IN MEMORY (NOT STACK)
                        ST=10       RESULT PUSHED ONTO STACK
                        ST=11       RESULT OUTPUT

                        DST = 2-BYTE RESULT STORED IN MEMORY (NOT STACK)
```

```
            $EJECT

429   2             DECLARE DEFTBL(256) ADDRESS DATA(

            /*00*/  0000H,0C000H, 0000H,0C000H, 8080H, 8080H, 8080H, 0040H,
                    0008H, 0340H, 0100H,0C000H, 4080H, 4080H, 4000H, 0040H,
            /*10*/  0008H, 3000H, 0000H, 3000H, 2080H, 2080H, 2000H, 0040H,
                    0008H, 0340H, 0100H, 3000H, 1080H, 1080H, 1000H, 0040H,
            /*20*/  0008H, 0C00H, 0000H, 0C00H, 0880H, 0880H, 0800H, 0000H,
                    0008H, 0340H, 0C00H, 0C00H, 0480H, 0480H, 0400H, 0000H,
            /*30*/  0008H, 0200H, 0000H, 0200H, 0080H, 0080H, 0000H, 0040H,
                    0008H, 0340H, 0100H, 0100H, 0180H, 0180H, 0100H, 0000H,
            /*40*/  0000H, 8000H, 8000H, 8000H, 8000H, 8000H, 8000H, 8000H,
                    4000H, 0000H, 4000H, 4000H, 4000H, 4000H, 4000H, 4000H,
            /*50*/  2000H, 2000H, 0000H, 2000H, 2000H, 2000H, 2000H, 2000H,
                    1000H, 1000H, 1000H, 0000H, 1000H, 1000H, 1000H, 1000H,
            /*60*/  0800H, 0800H, 0800H, 0800H, 0000H, 0800H, 0800H, 0800H,
                    0400H, 0400H, 0400H, 0400H, 0400H, 0000H, 0400H, 0400H,
            /*70*/  0000H, 0000H, 0000H, 0000H, 0000H, 0000H, 0008H, 0000H,
                    0100H, 0100H, 0100H, 0100H, 0100H, 0100H, 0100H, 0000H,
            /*80*/  00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H,
                    00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H,
            /*90*/  00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 01C0H,
                    00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H,
            /*A0*/  00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H,
                    00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 01C0H,
            /*B0*/  00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H,
                    00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H, 00C0H,
            /*C0*/  0000H,0C000H, 0000H, 0000H, 0000H, 0000H, 00C0H, 0000H,
                    0000H, 0000H, 0000H, 0008H, 0000H, 0000H, 00C0H, 0000H,
            /*D0*/  0000H, 3000H, 0000H, 0000H, 0000H, 0000H, 00C0H, 0000H,
                    0000H, 0008H, 0000H, 0100H, 0000H, 0008H, 00C0H, 0000H,
            /*E0*/  0000H, 0C00H, 0000H, 0C00H, 0000H, 0000H, 00C0H, 0000H,
                    0000H, 0000H, 0000H, 3C00H, 0000H, 0008H, 00C0H, 0000H,
            /*F0*/  0000H, 01C0H, 0000H, 0008H, 0000H, 0000H, 00C0H, 0000H,
                    0000H, 0200H, 0000H, 0008H, 0000H, 0008H, 00C0H, 0000H);

            /*
              V(5)  ! SZP  !  CY  !  PC  ! DB/R ! INTE ! OFL  ! DB/W ! ADDR !
              V(6)  !  B   !  C   !  D   !  E   !  H   !  L   !  SP  !  A   !

                        V(5) ::= LOW(DEF)
                        V(6) ::= HIGH(DEF)
```

```
                $EJECT

430   2         MEM:    PROCEDURE(ADDR) BYTE;
431   3                 DECLARE ADDR ADDRESS;
432   3                 IF .MEMORY+ADDR > 0F6B0H
                          THEN DO;
434   4                       CALL ALPHA(LP,52,.('*** ATTEMPT TO ACCE',
                                'SS OUT OF BOUNDS MEMORY LOCATION.'));
435   4                       CALL EOL;
436   4                       CALL EXIT;
437   4                       END;
438   3                 LASTMEMTOQ80 = MEMORY(ADDR);
439   3                 RETURN LASTMEMTOQ80;
440   3                 END MEM;
441   2         MEMW:   PROCEDURE(ADDR,VAL);
442   3                 DECLARE ADDR ADDRESS;
443   3                 DECLARE VAL BYTE;
444   3                 IF .MEMORY+ADDR > 0F6B0H
                          THEN DO;
446   4                       CALL ALPHA(LP,51,.('*** ATTEMPT TO WRITE',
                                ' OUT OF BOUNDS MEMORY LOCATION.'));
447   4                       CALL EOL;
448   4                       CALL EXIT;
449   4                       END;
450   3                 MEMORY(ADDR) = VAL;
451   3                 END MEMW;
```

```
                $EJECT

452   2     PRINT:  PROCEDURE;
453   3             DECLARE T BYTE;

454   3             CALL ALPHA(LP,1,.('!'));
455   3             IF EQUBIT(.OLDDEF,2,1,1B)
                      THEN DO;
457   4                       CALL HEX  (LP,2,.PC+1);
458   4                       CALL HEX  (LP,2,.PC);
459   4                       END;
460   3                 ELSE CALL SPACE(LP,4);
461   3             CALL ALPHA(LP,1,.('!'));
462   3             IF EQUBIT(.OLDDEF,3,1,1B)
                      THEN CALL HEX  (LP,2,.LASTMEMTOQ80);
464   3                 ELSE CALL SPACE(LP,2);
465   3             CALL SPACE(LP,1);
466   3             T = SHR(V(3),3) AND 10H;
467   3             CALL HEX  (LP,1,.T);
468   3             CALL SPACE(LP,1);
469   3             T = SHR(V(3),2) AND 10H;
470   3             CALL HEX  (LP,1,.T);
471   3             CALL SPACE(LP,1);
472   3             T = SHL(V(3),1) AND 10H;
473   3             CALL HEX  (LP,1,.T);
474   3             CALL SPACE(LP,1);
475   3             T = SHL(V(3),3) AND 10H;
476   3             CALL HEX  (LP,1,.T);
477   3             CALL ALPHA(LP,1,.('!'));
478   3             IF EQUBIT(.OLDDEF,7,1,1B)
                      THEN DO;
480   4                       CALL HEX  (LP,2,.OLDV(1));
481   4                       CALL HEX  (LP,2,.OLDV(0));
482   4                       END;
483   3                 ELSE CALL SPACE(LP,4);
484   3             CALL SPACE(LP,1);
485   3             IF EQUBIT(.OLDDEF,6,1,1B)
                      THEN CALL HEX  (LP,2,.OLDV(2));
487   3                 ELSE CALL SPACE(LP,2);
488   3             CALL SPACE(LP,1);
489   3             T = SHR(OLDV(4),1) AND 10H;
490   3             CALL HEX  (LP,1,.T);
491   3             CALL SPACE(LP,1);
492   3             T = OLDV(4) AND 10H;
493   3             CALL HEX  (LP,1,.T);
494   3             CALL SPACE(LP,1);
495   3             T = SHL(OLDV(4),1) AND 10H;
496   3             CALL HEX  (LP,1,.T);
497   3             CALL SPACE(LP,1);
498   3             T = SHL(OLDV(4),2) AND 10H;
499   3             CALL HEX  (LP,1,.T);
500   3             CALL SPACE(LP,1);
501   3             T = SHL(OLDV(4),3) AND 10H;
502   3             CALL HEX  (LP,1,.T);
503   3             CALL SPACE(LP,1);
504   3             T = SHL(OLDV(4),4) AND 10H;
505   3             CALL HEX  (LP,1,.T);
```

```
506   3              CALL SPACE(LP,1);
507   3              IF EQUBIT(.OLDDEF,4,1,1B)
                        THEN DO;
509   4                      T = SHR(OLDV(3),1) AND 10H;
510   4                      CALL HEX  (LP,1,.T);
511   4                      END;
512   3                 ELSE CALL SPACE(LP,1);
513   3              CALL SPACE(LP,1);
514   3              T = OLDV(3) AND 10H;
515   3              CALL HEX  (LP,1,.T);
516   3              CALL SPACE(LP,1);
517   3              T = SHL(OLDV(3),2) AND 10H;
518   3              CALL HEX  (LP,1,.T);
519   3              CALL SPACE(LP,1);
520   3              T = SHL(V(3),4) AND 10H;
521   3              CALL HEX  (LP,1,.T);
522   3              CALL ALPHA(LP,1,.('!'));
523   3              IF EQUBIT(.OLDDEF,8,1,1B)
                        THEN CALL HEX  (LP,2,.BC+1);
525   3                 ELSE CALL SPACE(LP,2);
526   3              IF EQUBIT(.OLDDEF,9,1,1B)
                        THEN CALL HEX  (LP,2,.BC);
528   3                 ELSE CALL SPACE(LP,2);
529   3              CALL SPACE(LP,1);
530   3              IF EQUBIT(.OLDDEF,10,1,1B)
                        THEN CALL HEX  (LP,2,.DE+1);
532   3                 ELSE CALL SPACE(LP,2);
533   3              IF EQUBIT(.OLDDEF,11,1,1B)
                        THEN CALL HEX  (LP,2,.DE);
535   3                 ELSE CALL SPACE(LP,2);
536   3              CALL SPACE(LP,1);
537   3              IF EQUBIT(.OLDDEF,12,1,1B)
                        THEN CALL HEX  (LP,2,.HL+1);
539   3                 ELSE CALL SPACE(LP,2);
540   3              IF EQUBIT(.OLDDEF,13,1,1B)
                        THEN CALL HEX  (LP,2,.HL);
542   3                 ELSE CALL SPACE(LP,2);
543   3              CALL SPACE(LP,1);
544   3              IF EQUBIT(.OLDDEF,14,1,1B)
                        THEN DO;
546   4                      CALL HEX  (LP,2,.SP+1);
547   4                      CALL HEX  (LP,2,.SP);
548   4                      END;
549   3                 ELSE CALL SPACE(LP,4);
550   3              CALL SPACE(LP,1);
551   3              IF EQUBIT(.OLDDEF,15,1,1B)
                        THEN CALL HEX  (LP,2,.A);
553   3                 ELSE CALL SPACE(LP,2);
554   3              CALL ALPHA(LP,1,.('!'));
555   3              IF EQUBIT(.OLDDEF,0,1,1B)
                        THEN DO;
557   4                      T = S AND 10H;
558   4                      CALL HEX  (LP,1,.T);
559   4                      CALL SPACE(LP,1);
560   4                      T = Z AND 10H;
561   4                      CALL HEX  (LP,1,.T);
562   4                      CALL SPACE(LP,1);
```

```
563   4                       T = P AND 10H;
564   4                       CALL HEX  (LP,1,.T);
565   4                       CALL SPACE(LP,1);
566   4                       END;
567   3                    ELSE CALL SPACE(LP,6);
568   3                 IF EQUBIT(.OLDDEF,1,1,1B)
                          THEN DO;
570   4                       T = CY AND 10H;
571   4                       CALL HEX  (LP,1,.T);
572   4                       CALL SPACE(LP,1);
573   4                       END;
574   3                    ELSE CALL SPACE(LP,2);
575   3                 IF EQUBIT(.OLDDEF,5,1,1B)
                          THEN DO;
577   4                       T = OFL AND 10H;
578   4                       CALL HEX  (LP,1,.T);
579   4                       END;
580   3                    ELSE CALL SPACE(LP,1);
581   3                 CALL ALPHA(LP,1,.('!'));
582   3                 CALL EOL;
583   3                 END PRINT;
```

```
                $EJECT

 584   2              SCHEDULE:PROCEDURE(STATE);
 585   3                      DECLARE STATE BYTE;
 586   3                      DECLARE (I,J) BYTE;
 587   3                      DECLARE RHHI BYTE;

 588   3                      CALL MOVE(7,.V,.OLDV);

 589   3                      DO I = 0 TO 49;
 590   4                      IF (COMM(I).LOC = INSTRADDR) AND
                                ((STATE = 0) OR (STATE = 13))
                                THEN COMM(I).COUNT = COMM(I).COUNT + 1;
 592   4                      END;

 593   3                      I = 0;
 594   3                      DO WHILE (COMM(I).LOC <> INSTRADDR) OR
                                       (COMM(I).COUNT <> COMM(I).TRIGGER);
 595   4                      I = I + 1;
 596   4                      IF (I = 50) OR (COMM(I).TRIGGER = 0)
                                THEN DO;
 598   5                           RESIN, INT, HLT, HOLD = 0;
 599   5                           CALL SETBIT(.V,24,2,00B);
 600   5                           CALL SETBIT(.V,28,1,0B);
 601   5                           CALL SETBIT(.V,30,1,0B);
 602   5                           RETURN;
 603   5                           END;
 604   4                      END;
 605   3                      IF (STATE = 0) OR (STATE = 13)
                                THEN SEQUCOUNT = 0;
 607   3                      IF STATE < 16
                                THEN DO;
 609   4                           RHHI = COMM(I).FIXED(SHR(STATE,1));
 610   4                           IF STATE AND 01H
                                       THEN RHHI = RHHI AND 0FH;
 612   4                                  ELSE RHHI = SHR(RHHI,4);
 613   4                           END;
 614   3                         ELSE DO;
 615   4                           RHHI = COMM(I).SEQU(SHR(SEQUCOUNT,1));
 616   4                           IF SEQUCOUNT AND 01H
                                       THEN RHHI = RHHI AND 0FH;
 618   4                                  ELSE RHHI = SHR(RHHI,4);
 619   4                           SEQUCOUNT = SEQUCOUNT + 1;
 620   4                           END;
 621   3                      IF RHHI AND 01H
                                THEN DO;
 623   4                           INT = 0FFH;
 624   4                           CALL SETBIT(.V,25,1,1B);
 625   4                           CALL MOVE(3,.COMM(I).JAM,.INTJAMWORD);
 626   4                           END;
 627   3                         ELSE DO;
 628   4                           INT = 0;
 629   4                           CALL SETBIT(.V,25,1,0B);
 630   4                           END;
 631   3                      IF (RHHI AND 02H) = 02H
                                THEN DO;
 633   4                           HOLD = 0FFH;
```

```
634   4                              CALL SETBIT(.V,28,1,1B);
635   4                              END;
636   3                         ELSE DO;
637   4                              HOLD = 0;
638   4                              CALL SETBIT(.V,28,1,0B);
639   4                              END;
640   3                    IF (RHHI AND 04H) = 04H
                           THEN DO;
642   4                              HLT = 0FFH;
643   4                              CALL SETBIT(.V,30,1,1B);
644   4                              END;
645   3                         ELSE DO;
646   4                              HLT = 0;
647   4                              CALL SETBIT(.V,30,1,0B);
648   4                              END;
649   3                    IF (RHHI AND 08H) = 08H
                           THEN DO;
651   4                              RESIN = 0FFH;
652   4                              CALL SETBIT(.V,24,1,1B);
653   4                              END;
654   3                         ELSE DO;
655   4                              RESIN = 0;
656   4                              CALL SETBIT(.V,24,1,0B);
657   4                              END;
658   3                    END SCHEDULE;
```

```
$EJECT

                        /****************/
                        /*  START-UP    */
                        /****************/

659   2             CALL MOVE(7,.(0,0,0,80H,0,0,0),.V);
660   2             CALL MOVE(7,.(0,0,0,0,0,0,0),.OLDV);
661   2             CALL PRINT;
```

```
                $EJECT

                                /************/
                                /*  RESET   */
                                /************/

662   2         RESET:  DO;
663   3                 PC = 0;
664   3                 INSTRADDR = 0;
665   3                 V(0) = 0FFH;
666   3                 V(1) = 0FFH;
667   3                 V(2) = 0FFH;
668   3                 CALL SETBIT(.V,26,1,INTE);
669   3                 CALL SETBIT(.V,27,1,0B);
670   3                 CALL SETBIT(.V,29,1,0B);
671   3                 CALL SETBIT(.V,31,1,0B);
672   3                 CALL SETBIT(.V,33,1,OFL);
673   3                 CALL SETBIT(.V,34,6,000000B);
674   3                 DEF = OLDDEF AND 0FFEFH OR 0023H;

675   3                 CALL ALPHA(80,6,.V);
676   3                 CALL SCHEDULE(16);
677   3                 CALL PRINT;

678   3                 IF RESIN THEN GOTO RESET;
680   3                          ELSE GOTO IFCH1;
681   3                 END;
```

```
                $EJECT

                                /************/
                                /*  IFCH1   */
                                /************/

682   2         IFCH1:  DO;
683   3                 V(0) = LOW (PC);
684   3                 V(1) = HIGH(PC);
685   3                 V(2) = LASTMEMTOQ80;
686   3                 CALL SETBIT(.V,26,1,INTE);
687   3                 CALL SETBIT(.V,27,1,0B);
688   3                 CALL SETBIT(.V,29,1,0B);
689   3                 CALL SETBIT(.V,31,1,0B);
690   3                 CALL SETBIT(.V,33,1,OFL);
691   3                 CALL SETBIT(.V,34,6,010010B);
692   3                 DEF = OLDDEF AND 0FFFDH OR 0011H;

693   3                 CALL ALPHA(80,6,.V);
694   3                 INSTRADDR = PC;
695   3                 CALL SCHEDULE(0);
696   3                 OPCODE = MEM(PC);
697   3                 CALL PRINT;

698   3                 PC = PC + 1;
699   3                 OFL = 0;
700   3                 OLDDEF = OLDDEF OR 0004H;

701   3                 IF RESIN THEN GOTO RESET;
703   3                 IF (OPCODE AND 0C7H) = 0C0H THEN GOTO COND;
705   3                 IF (TBL(OPCODE) AND 0C0H) > 0 THEN GOTO IFCH2;
707   3                 DO CASE SHR(TBL(OPCODE) AND 30H,4);
708   4                         GOTO EXEC;        /* OF = 00 */
709   4                         GOTO OFCH1;       /* OF = 01 */
710   4                         GOTO POP1;        /* OF = 10 */
711   4                         GOTO INPUT;       /* OF = 11 */
712   4                 END;
713   3                 END;
```

```
                $EJECT
                                /************/
                                /*  IFCH2  */
                                /************/

714   2         IFCH2:  DO;
715   3                 V(0) = LOW (PC);
716   3                 V(1) = HIGH(PC);
717   3                 V(2) = LASTMEMTOQ80;
718   3                 CALL SETBIT(.V,26,1,INTE);
719   3                 CALL SETBIT(.V,27,1,0B);
720   3                 CALL SETBIT(.V,29,1,0B);
721   3                 CALL SETBIT(.V,31,1,0B);
722   3                 CALL SETBIT(.V,33,1,0B);
723   3                 CALL SETBIT(.V,34,6,010010B);
724   3                 DEF = OLDDEF;

725   3                 CALL ALPHA(80,6,.V);
726   3                 CALL SCHEDULE(1);
727   3                 DATABYTE = MEM(PC);
728   3                 CALL PRINT;

729   3                 PC = PC + 1;

730   3                 IF RESIN THEN GOTO RESET;
732   3                 IF (TBL(OPCODE) AND 80H) = 80H THEN GOTO IFCH3;
734   3                 DO CASE SHR(TBL(OPCODE) AND 30H,4);
735   4                         GOTO EXEC;        /* OF = 00 */
736   4                         GOTO OFCH1;       /* OF = 01 */
737   4                         GOTO POP1;        /* OF = 10 */
738   4                         GOTO INPUT;       /* OF = 11 */
739   4                 END;
740   3                 END;
```

```
                $EJECT
                                /************/
                                /*  IFCH3   */
                                /************/

741   2         IFCH3:  DO;
742   3                 V(0) = LOW (PC);
743   3                 V(1) = HIGH(PC);
744   3                 V(2) = LASTMEMTOQ80;
745   3                 CALL SETBIT(.V,26,1,INTE);
746   3                 CALL SETBIT(.V,27,1,0B);
747   3                 CALL SETBIT(.V,29,1,0B);
748   3                 CALL SETBIT(.V,31,1,0B);
749   3                 CALL SETBIT(.V,33,1,0B);
750   3                 CALL SETBIT(.V,34,6,010010B);
751   3                 DEF = OLDDEF;

752   3                 CALL ALPHA(80,6,.V);
753   3                 CALL SCHEDULE(2);
754   3                 ADDRH = MEM(PC);
755   3                 CALL PRINT;

756   3                 PC = PC + 1;

757   3                 IF RESIN THEN GOTO RESET;
759   3                 DO CASE SHR(TBL(OPCODE) AND 30H,4);
760   4                         GOTO EXEC;        /* OF = 00 */
761   4                         GOTO OFCH1;       /* OF = 01 */
762   4                         GOTO POP1;        /* OF = 10 */
763   4                         GOTO INPUT;       /* OF = 11 */
764   4                 END;
765   3                 END;
```

```
                $EJECT

                                /**********/
                                /*  COND  */
                                /**********/

766   2         COND:   DO;
767   3                 DECLARE T BYTE;

768   3                 DO CASE SHR(OPCODE AND 38H, 3);
769   4                 T = (Z = 0);
770   4                 T = (Z > 0);
771   4                 T = (CY = 0);
772   4                 T = (CY > 0);
773   4                 T = (P = 0);
774   4                 T = (P > 0);
775   4                 T = (S = 0);
776   4                 T = (S > 0);
777   4                 END;

778   3                 IF T THEN GOTO POP1;
780   3                      ELSE GOTO EXEC;

781   3                 END;
```

```
                $EJECT
                                /************/
                                /*  OFCH1   */
                                /************/

782   2         OFCH1:  DO;
783   3                 IF OPCODE = 0AH THEN CALL MOVE(2,.BC,.V);
785   3                 IF OPCODE = 1AH THEN CALL MOVE(2,.DE,.V);
787   3                 IF OPCODE = 2AH THEN CALL MOVE(2,.ADDR,.V);
789   3                 IF OPCODE = 3AH THEN CALL MOVE(2,.ADDR,.V);
791   3                 IF (OPCODE AND 0CFH) <> 0AH THEN CALL MOVE(2,.HL,.V);
793   3                 V(2) = LASTMEMTOQ80;
794   3                 CALL SETBIT(.V,26,1,INTE);
795   3                 CALL SETBIT(.V,27,1,0B);
796   3                 CALL SETBIT(.V,29,1,0B);
797   3                 CALL SETBIT(.V,31,1,0B);
798   3                 CALL SETBIT(.V,33,1,0B);
799   3                 CALL SETBIT(.V,34,6,000010B);
800   3                 DEF = OLDDEF;

801   3                 CALL ALPHA(80,6,.V);
802   3                 CALL SCHEDULE(3);
803   3                 MDATA = MEM(MADDR);
804   3                 CALL PRINT;

805   3                 IF RESIN THEN GOTO RESET;
807   3                 IF (TBL(OPCODE) AND 08H) = 08H THEN GOTO OFCH2;
809   3                                                ELSE GOTO EXEC;
810   3                 END;
```

```
                $EJECT
                                        /************/
                                        /*  OFCH2   */
                                        /************/

811   2         OFCH2:  DO;
812   3                 V(0) = LOW (MADDR+1);
813   3                 V(1) = HIGH(MADDR+1);
814   3                 V(2) = LASTMEMTOQ80;
815   3                 CALL SETBIT(.V,26,1,INTE);
816   3                 CALL SETBIT(.V,27,1,0B);
817   3                 CALL SETBIT(.V,29,1,0B);
818   3                 CALL SETBIT(.V,31,1,0B);
819   3                 CALL SETBIT(.V,33,1,0B);
820   3                 CALL SETBIT(.V,34,6,000010B);
821   3                 DEF = OLDDEF;

822   3                 CALL ALPHA(80,6,.V);
823   3                 CALL SCHEDULE(4);
824   3                 MDATAH = MEM(ADDR+1);
825   3                 CALL PRINT;

826   3                 IF RESIN THEN GOTO RESET;
828   3                          ELSE GOTO EXEC;
829   3                 END;
```

```
                $EJECT
                                /**********/
                                /*  POP1  */
                                /**********/

830   2         POP1:   DO;
831   3                 V(0) = LOW (SP);
832   3                 V(1) = HIGH(SP);
833   3                 V(2) = LASTMEMTOQ80;
834   3                 CALL SETBIT(.V,26,1,INTE);
835   3                 CALL SETBIT(.V,27,1,0B);
836   3                 CALL SETBIT(.V,29,1,0B);
837   3                 CALL SETBIT(.V,31,1,0B);
838   3                 CALL SETBIT(.V,33,1,0B);
839   3                 CALL SETBIT(.V,34,6,100010B);
840   3                 DEF = OLDDEF;

841   3                 CALL ALPHA(80,6,.V);
842   3                 CALL SCHEDULE(5);
843   3                 MDATA = MEM(SP);
844   3                 CALL PRINT;

845   3                 SP = SP + 1;

846   3                 IF RESIN THEN GOTO RESET;
848   3                         ELSE GOTO POP2;
849   3                 END;
```

```
                $EJECT

                                /***********/
                                /*  POP2   */
                                /***********/

850   2         POP2:   DO;
851   3                 V(0) = LOW (SP);
852   3                 V(1) = HIGH(SP);
853   3                 V(2) = LASTMEMTOQ80;
854   3                 CALL SETBIT(.V,26,1,INTE);
855   3                 CALL SETBIT(.V,27,1,0B);
856   3                 CALL SETBIT(.V,29,1,0B);
857   3                 CALL SETBIT(.V,31,1,0B);
858   3                 CALL SETBIT(.V,33,1,0B);
859   3                 CALL SETBIT(.V,34,6,100010B);
860   3                 DEF = OLDDEF;

861   3                 CALL ALPHA(80,6,.V);
862   3                 CALL SCHEDULE(6);
863   3                 MDATAH = MEM(SP);
864   3                 CALL PRINT;

865   3                 SP = SP + 1;

866   3                 IF RESIN THEN GOTO RESET;
868   3                          ELSE GOTO EXEC;
869   3                 END;
```

```
                $EJECT
                                /************/
                                /*  INPUT   */
                                /************/

870   2         INPUT:  DO;
871   3                 DECLARE (L,S) ADDRESS;
872   3                 V(0) = DATABYTE;
873   3                 V(1) = 0;
874   3                 V(2) = LASTMEMTOQ80;
875   3                 CALL SETBIT(.V,26,1,INTE);
876   3                 CALL SETBIT(.V,27,1,0B);
877   3                 CALL SETBIT(.V,29,1,0B);
878   3                 CALL SETBIT(.V,31,1,0B);
879   3                 CALL SETBIT(.V,33,1,0B);
880   3                 CALL SETBIT(.V,34,6,001000B);
881   3                 DEF = OLDDEF;

882   3                 CALL ALPHA(80,6,.V);
883   3                 CALL SCHEDULE(7);

884   3                 CALL READ(Q80IN,.MDATA,1,.L,.S);
885   3                 IF (L=0) OR (INSTATUS=13)
                             THEN DO;
887   4                       CALL ALPHA(LP,43,.('*** ATTEMPT TO READ PAST END',
                                                  ' OF INPUT FILE.'));
888   4                       CALL EOL;
889   4                       CALL EXIT;
890   4                       END;
891   3                 LASTMEMTOQ80 = MDATA;

892   3                 CALL PRINT;

893   3                 IF RESIN THEN GOTO RESET;
895   3                          ELSE GOTO EXEC;
896   3                 END;
```

```
                $EJECT
                                /**********/
                                /*  EXEC  */
                                /**********/

897   2         EXEC:   DO;
898   3                 DECLARE (DUMMY, ADD, SUB) BYTE;
899   3                 DOFL:   PROCEDURE(OP1, OP2) BYTE;
900   4                         DECLARE (OP1, OP2, RES) ADDRESS;
901   4                         RES = OP1 + OP2;
902   4                         IF ROL(OP1 AND OP2 AND NOT RES, 1) OR
                                   ROL(NOT OP1 AND NOT OP2 AND RES, 1)
                                   THEN RETURN 0FFH;
904   4                            ELSE RETURN 0;
905   4                         END DOFL;

906   3                 OFLCY:  PROCEDURE(OP1, OP2, CB, OP);
907   4                         DECLARE (OP1, OP2, CB, OP) BYTE;
908   4                         DECLARE (A, B, C) ADDRESS;
909   4                         A = OP1;
910   4                         B = OP2;
911   4                         IF OP=ADD THEN DO;
913   5                                        C = A + B + (CB AND 01H);
914   5                                        CY = NOT (HIGH(C) = 0);
915   5                                        END;
916   4                                  ELSE DO;
917   5                                        C = A - B - (CB AND 01H);
918   5                                        CY = (HIGH(C) = 0);
919   5                                        END;
920   4                         IF ROL(OP1, 1) THEN A = A OR 0FF00H;
922   4                         IF ROL(OP2, 1) THEN B = B OR 0FF00H;
924   4                         IF OP=ADD THEN C = A + B + (CB AND 01H);
926   4                                   ELSE C = A - B - (CB AND 01H);
927   4                         OFL = ((C AND 0FF80H) <> 0FF80H) AND
                                        ((C AND 0FF80H) <>  0000H);
928   4                         END OFLCY;

929   3                 ADD = 0;
930   3                 SUB = 1;
931   3                 OLDDEF = OLDDEF OR DEFTBL(OPCODE);
```

```
                $EJECT

932   3             DO CASE OPCODE;

                /* 00: NOOP */
933   4             ;

                /* 01: LXI B,DATA16 */
934   4             BC = ADDR;
                /* 02: STAX B */
935   4             SDATA = A;

                /* 03: INX B */
936   4             BC = BC + 1;

                /* 04: INR B */
937   4             DO;
938   5             DUMMY = B + 1;
939   5             S = SIGN;
940   5             DUMMY = B + 1;
941   5             Z = ZERO;
942   5             B = B + 1;
943   5             P = PARITY;
944   5             END;

                /* 05: DCR B */
945   4             DO;
946   5             DUMMY = B - 1;
947   5             S = SIGN;
948   5             DUMMY = B - 1;
949   5             Z = ZERO;
950   5             B = B - 1;
951   5             P = PARITY;
952   5             END;

                /* 06: MVI B,DATA8 */
953   4             B = DATABYTE;

                /* 07: RLC */
954   4             DO;
955   5             A = ROL(A,1);
956   5             CY = CARRY;
957   5             END;

                /* 08: HALT */
958   4             DO;
959   5             INTE = 0FFH;
960   5             IF HLT AND HOLD THEN GOTO IEHHOLD;
962   5             IF HLT          THEN GOTO HALTIE;
964   5             IF HOLD         THEN GOTO IHHOLD;
966   5             IF INT AND INTE THEN GOTO INTJAM1;
968   5                             ELSE GOTO HALTI;
969   5             END;

                /* 09: DAD B */
970   4             DO;
971   5             OFL = DOFL(HL,BC);
```

```
 972   5              HL = HL + BC;
 973   5              CY = CARRY;
 974   5              END;

                /* 0A: LDAX B */
 975   4              A = MDATA;

                /* 0B: DCX B */
 976   4              BC = BC - 1;

                /* 0C: INR C */
 977   4              DO;
 978   5              DUMMY = C + 1;
 979   5              S = SIGN;
 980   5              DUMMY = C - 1;
 981   5              Z = ZERO;
 982   5              C = C - 1;
 983   5              P = PARITY;
 984   5              END;

                /* 0D: DCR C */
 985   4              DO;
 986   5              DUMMY = C - 1;
 987   5              S = SIGN;
 988   5              DUMMY = C - 1;
 989   5              Z = ZERO;
 990   5              C = C - 1;
 991   5              P = PARITY;
 992   5              END;

                /* 0E: MVI C,DATA8 */
 993   4              C = DATABYTE;

                /* 0F: RRC */
 994   4              DO;
 995   5              A = ROR(A,1);
 996   5              CY = CARRY;
 997   5              END;

                /* 10: HALT */
 998   4              DO;
 999   5              INTE = 0FFH;
1000   5              IF HLT AND HOLD THEN GOTO IEHHOLD;
1002   5              IF HLT          THEN GOTO HALTIE;
1004   5              IF HOLD         THEN GOTO IHHOLD;
1006   5              IF INT AND INTE THEN GOTO INTJAM1;
1008   5                              ELSE GOTO HALTI;
1009   5              END;

                /* 11: LXI D,DATA16 */
1010   4              DE = ADDR;

                /* 12: STAX D */
1011   4              SDATA = A;

                /* 13: INX D */
1012   4              DE = DE + 1;
```

```
                /* 14: INR D */
1013    4               DO;
1014    5               DUMMY = D + 1;
1015    5               S = SIGN;
1016    5               DUMMY = D + 1;
1017    5               Z = ZERO;
1018    5               D = D + 1;
1019    5               P = PARITY;
1020    5               END;

                /* 15: DCR D */
1021    4               DO;
1022    5               DUMMY = D - 1;
1023    5               S = SIGN;
1024    5               DUMMY = D - 1;
1025    5               Z = ZERO;
1026    5               D = D - 1;
1027    5               P = PARITY;
1028    5               END;

                /* 16: MVI D,DATA8 */
1029    4               D = DATABYTE;

                /* 17: RAL */
1030    4               DO;
1031    5               DUMMY = 0FFH + (CY AND 01H);
1032    5               A = SCL(A,1);
1033    5               CY = CARRY;
1034    5               END;

                /* 18: HALT */
1035    4               DO;
1036    5               INTE = 0FFH;
1037    5               IF HLT AND HOLD THEN GOTO IEHHOLD;
1039    5               IF HLT          THEN GOTO HALTIE;
1041    5               IF HOLD         THEN GOTO IHHOLD;
1043    5               IF INT AND INTE THEN GOTO INTJAM1;
1045    5                                ELSE GOTO HALTI;
1046    5               END;

                /* 19: DAD D */
1047    4               DO;
1048    5               OFL = DOFL(HL,DE);
1049    5               HL = HL + DE;
1050    5               CY = CARRY;
1051    5               END;

                /* 1A: LDAX D */
1052    4               A = MDATA;

                /* 1B: DCX D */
1053    4               DE = DE - 1;

                /* 1C: INR E */
1054    4               DO;
1055    5               DUMMY = E + 1;
```

```
1056   5              S = SIGN;
1057   5              DUMMY = E + 1;
1058   5              Z = ZERO;
1059   5              E = E + 1;
1060   5              P = PARITY;
1061   5              END;

               /* 1D: DCR E */
1062   4              DO;
1063   5              DUMMY = E - 1;
1064   5              S = SIGN;
1065   5              DUMMY = E - 1;
1066   5              Z = ZERO;
1067   5              E = E - 1;
1068   5              P = PARITY;
1069   5              END;

               /* 1E: MVI E,DATA8 */
1070   4              E = DATABYTE;

               /* 1F: RAR */
1071   4              DO;
1072   5              DUMMY = 0FFH + (CY AND 01H);
1073   5              A = SCR(A,1);
1074   5              CY = CARRY;
1075   5              END;

               /* 20: HALT */
1076   4              DO;
1077   5              INTE = 0FFH;
1078   5              IF HLT AND HOLD THEN GOTO IEHHOLD;
1080   5              IF HLT          THEN GOTO HALTIE;
1082   5              IF HOLD         THEN GOTO IHHOLD;
1084   5              IF INT AND INTE THEN GOTO INTJAM1;
1086   5                              ELSE GOTO HALTI;
1087   5              END;

               /* 21: LXI H,DATA16 */
1088   4              HL = ADDR;

               /* 22: SHLD ADDR */
1089   4              SDATA2 = HL;

               /* 23: INX H */
1090   4              HL = HL + 1;

               /* 24: INR H */
1091   4              DO;
1092   5              DUMMY = H + 1;
1093   5              S = SIGN;
1094   5              DUMMY = H + 1;
1095   5              Z = ZERO;
1096   5              H = H + 1;
1097   5              P = PARITY;
1098   5              END;

               /* 25: DCR H */
```

```
1099   4                DO;
1100   5                DUMMY = H - 1;
1101   5                S = SIGN;
1102   5                DUMMY = H - 1;
1103   5                Z = ZERO;
1104   5                H = H - 1;
1105   5                P = PARITY;
1106   5                END;

                  /* 26: MVI H,DATA8 */
1107   4                H = DATABYTE;

                  /* 27: DAA IMPLEMENTED AS SYNC, LOOKS LIKE NOOP */
1108   4                ;

                  /* 28: HALT */
1109   4                DO;
1110   5                INTE = 0FFH;
1111   5                IF HLT AND HOLD THEN GOTO IEHHOLD;
1113   5                IF HLT          THEN GOTO HALTIE;
1115   5                IF HOLD         THEN GOTO IHHOLD;
1117   5                IF INT AND INTE THEN GOTO INTJAM1;
1119   5                                ELSE GOTO HALTI;
1120   5                END;

                  /* 29: DAD H */
1121   4                DO;
1122   5                OFL = DOFL(HL,HL);
1123   5                HL = HL + HL;
1124   5                CY = CARRY;
1125   5                END;

                  /* 2A: LHLD ADDR */
1126   4                HL = ADDR;

                  /* 2B: DCX H */
1127   4                HL = HL - 1;

                  /* 2C: INR L */
1128   4                DO;
1129   5                DUMMY = L + 1;
1130   5                S = SIGN;
1131   5                DUMMY = L + 1;
1132   5                Z = ZERO;
1133   5                L = L + 1;
1134   5                P = PARITY;
1135   5                END;

                  /* 2D: DCR L */
1136   4                DO;
1137   5                DUMMY = L - 1;
1138   5                S = SIGN;
1139   5                DUMMY = L - 1;
1140   5                Z = ZERO;
1141   5                L = L - 1;
1142   5                P = PARITY;
1143   5                END;
```

```
                /* 2E: MVI L,DATA8 */
1144    4               L = DATABYTE;

                /* 2F: CMA */
1145    4               A = NOT A;

                /* 30: HALT */
1146    4               DO;
1147    5               INTE = 0FFH;
1148    5               IF HLT AND HOLD THEN GOTO IEHHOLD;
1150    5               IF HLT          THEN GOTO HALTIE;
1152    5               IF HOLD         THEN GOTO IHHOLD;
1154    5               IF INT AND INTE THEN GOTO INTJAM1;
1156    5                               ELSE GOTO HALTI;
1157    5               END;

                /* 31: LXI SP,DATA16 */
1158    4               SP = ADDR;

                /* 32: STA ADDR */
1159    4               SDATA = A;

                /* 33: INX SP */
1160    4               SP = SP + 1;

                /* 34: INR M */
1161    4               DO;
1162    5               DUMMY = MDATA + 1;
1163    5               S = SIGN;
1164    5               DUMMY = MDATA + 1;
1165    5               Z = ZERO;
1166    5               SDATA = MDATA + 1;
1167    5               P = PARITY;
1168    5               END;

                /* 35: DCR M */
1169    4               DO;
1170    5               DUMMY = MDATA - 1;
1171    5               S = SIGN;
1172    5               DUMMY = MDATA - 1;
1173    5               Z = ZERO;
1174    5               SDATA = MDATA - 1;
1175    5               P = PARITY;
1176    5               END;

                /* 36: MVI M,DATA8 */
1177    4               SDATA = DATABYTE;

                /* 37: STC */
1178    4               CY = 0FFH;

                /* 38: HALT */
1179    4               DO;
1180    5               INTE = 0FFH;
1181    5               IF HLT AND HOLD THEN GOTO IEHHOLD;
1183    5               IF HLT          THEN GOTO HALTIE;
```

```
1185   5              IF HOLD          THEN GOTO IHHOLD;
1187   5              IF INT AND INTE THEN GOTO INTJAM1;
1189   5                               ELSE GOTO HALTI;
1190   5              END;

                    /* 39: DAD SP */
1191   4              DO;
1192   5              OFL = DOFL(HL,SP);
1193   5              HL = HL + SP;
1194   5              CY = CARRY;
1195   5              END;

                    /* 3A: LDA ADDR */
1196   4              A = MDATA;

                    /* 3B: DCX SP */
1197   4              SP = SP - 1;

                    /* 3C: INR A */
1198   4              DO;
1199   5              DUMMY = A + 1;
1200   5              S = SIGN;
1201   5              DUMMY = A + 1;
1202   5              Z = ZERO;
1203   5              A = A + 1;
1204   5              P = PARITY;
1205   5              END;

                    /* 3D: DCR A */
1206   4              DO;
1207   5              DUMMY = A - 1;
1208   5              S = SIGN;
1209   5              DUMMY = A - 1;
1210   5              Z = ZERO;
1211   5              A = A - 1;
1212   5              P = PARITY;
1213   5              END;

                    /* 3E: MVI A,DATA8 */
1214   4              A = DATABYTE;

                    /* 3F: CMC */
1215   4              CY = NOT CY;

                    /* 40: MOV B,B */
1216   4              B = B;

                    /* 41: MOV B,C */
1217   4              B = C;

                    /* 42: MOV B,D */
1218   4              B = D;

                    /* 43: MOV B,E */
1219   4              B = E;

                    /* 44: MOV B,H */
```

```
1220   4                B = H;

                  /* 45: MOV B,L */
1221   4                B = L;

                  /* 46: MOV B,M */
1222   4                B = MDATA;

                  /* 47: MOV B,A */
1223   4                B = A;

                  /* 48: MOV C,B */
1224   4                C = B;

                  /* 49: MOV C,C */
1225   4                C = C;

                  /* 4A: MOV C,D */
1226   4                C = D;

                  /* 4B: MOV C,E */
1227   4                C = E;

                  /* 4C: MOV C,H */
1228   4                C = H;

                  /* 4D: MOV C,L */
1229   4                C = L;

                  /* 4E: MOV C,M */
1230   4                C = MDATA;

                  /* 4F: MOV C,A */
1231   4                C = A;

                  /* 50: MOV D,B */
1232   4                D = B;

                  /* 51: MOV D,C */
1233   4                D = C;

                  /* 52: MOV D,D */
1234   4                D = D;

                  /* 53: MOV D,E */
1235   4                D = E;

                  /* 54: MOV D,H */
1236   4                D = H;

                  /* 55: MOV D,L */
1237   4                D = L;

                  /* 56: MOV D,M */
1238   4                D = MDATA;

                  /* 57: MOV D,A */
```

```
1239   4                D = A;

                  /* 58: MOV E,B */
1240   4                E = B;

                  /* 59: MOV E,C */
1241   4                E = C;

                  /* 5A: MOV E,D */
1242   4                E = D;

                  /* 5B: MOV E,E */
1243   4                E = E;

                  /* 5C: MOV E,H */
1244   4                E = H;

                  /* 5D: MOV E,L */
1245   4                E = L;

                  /* 5E: MOV E,M */
1246   4                E = MDATA;

                  /* 5F: MOV E,A */
1247   4                E = A;

                  /* 60: MOV H,B */
1248   4                H = B;

                  /* 61: MOV H,C */
1249   4                H = C;

                  /* 62: MOV H,D */
1250   4                H = D;

                  /* 63: MOV H,E */
1251   4                H = E;

                  /* 64: MOV H,H */
1252   4                H = H;

                  /* 65: MOV H,L */
1253   4                H = L;

                  /* 66: MOV H,M */
1254   4                H = MDATA;

                  /* 67: MOV H,A */
1255   4                H = A;

                  /* 68: MOV L,B */
1256   4                L = B;

                  /* 69: MOV L,C */
1257   4                L = C;

                  /* 6A: MOV L,D */
```

```
1258  4                 L = D;

                  /* 6B: MOV L,E */
1259  4                 L = E;

                  /* 6C: MOV L,H */
1260  4                 L = H;

                  /* 6D: MOV L,L */
1261  4                 L = L;

                  /* 6E: MOV L,M */
1262  4                 L = MDATA;

                  /* 6F: MOV L,A */
1263  4                 L = A;

                  /* 70: MOV M,B */
1264  4                 SDATA = B;

                  /* 71: MOV M,C */
1265  4                 SDATA = C;

                  /* 72: MOV M,D */
1266  4                 SDATA = D;

                  /* 73: MOV M,E */
1267  4                 SDATA = E;

                  /* 74: MOV M,H */
1268  4                 SDATA = H;

                  /* 75: MOV M,L */
1269  4                 SDATA = L;

                  /* 76: HALT */
1270  4                 DO;
1271  5                 INTE = 0FFH;
1272  5                 IF HLT AND HOLD THEN GOTO IEHHOLD;
1274  5                 IF HLT          THEN GOTO HALTIE;
1276  5                 IF HOLD         THEN GOTO IHHOLD;
1278  5                 IF INT AND INTE THEN GOTO INTJAM1;
1280  5                                 ELSE GOTO HALTI;
1281  5                 END;

                  /* 77: MOV M,A */
1282  4                 SDATA = A;

                  /* 78: MOV A,B */
1283  4                 A = B;

                  /* 79: MOV A,C */
1284  4                 A = C;

                  /* 7A: MOV A,D */
1285  4                 A = D;
```

```
                /* 7B: MOV A,E */
1286   4                A = E;

                /* 7C: MOV A,H */
1287   4                A = H;

                /* 7D: MOV A,L */
1288   4                A = L;

                /* 7E: MOV A,M */
1289   4                A = MDATA;

                /* 7F: MOV A,A */
1290   4                A = A;

                /* 80: ADD B */
1291   4                DO;
1292   5                CALL OFLCY(A,B,0,ADD);
1293   5                DUMMY = A + B;
1294   5                S = SIGN;
1295   5                DUMMY = A + B;
1296   5                Z = ZERO;
1297   5                A = A + B;
1298   5                P = PARITY;
1299   5                END;

                /* 81: ADD C */
1300   4                DO;
1301   5                CALL OFLCY(A,C,0,ADD);
1302   5                DUMMY = A + C;
1303   5                S = SIGN;
1304   5                DUMMY = A + C;
1305   5                Z = ZERO;
1306   5                A = A + C;
1307   5                P = PARITY;
1308   5                END;

                /* 82: ADD D */
1309   4                DO;
1310   5                CALL OFLCY(A,D,0,ADD);
1311   5                DUMMY = A + D;
1312   5                S = SIGN;
1313   5                DUMMY = A + D;
1314   5                Z = ZERO;
1315   5                A = A + D;
1316   5                P = PARITY;
1317   5                END;

                /* 83: ADD E */
1318   4                DO;
1319   5                CALL OFLCY(A,E,0,ADD);
1320   5                DUMMY = A + E;
1321   5                S = SIGN;
1322   5                DUMMY = A + E;
1323   5                Z = ZERO;
1324   5                A = A + E;
1325   5                P = PARITY;
```

```
1326   5                END;

                  /* 84: ADD H */
1327   4                DO;
1328   5                CALL OFLCY(A,H,0,ADD);
1329   5                DUMMY = A + H;
1330   5                S = SIGN;
1331   5                DUMMY = A + H;
1332   5                Z = ZERO;
1333   5                A = A + H;
1334   5                P = PARITY;
1335   5                END;

                  /* 85: ADD L */
1336   4                DO;
1337   5                CALL OFLCY(A,L,0,ADD);
1338   5                DUMMY = A + L;
1339   5                S = SIGN;
1340   5                DUMMY = A + L;
1341   5                Z = ZERO;
1342   5                A = A + L;
1343   5                P = PARITY;
1344   5                END;

                  /* 86: ADD M */
1345   4                DO;
1346   5                CALL OFLCY(A,MDATA,0,ADD);
1347   5                DUMMY = A + MDATA;
1348   5                S = SIGN;
1349   5                DUMMY = A + MDATA;
1350   5                Z = ZERO;
1351   5                A = A + MDATA;
1352   5                P = PARITY;
1353   5                END;

                  /* 87: ADD A */
1354   4                DO;
1355   5                CALL OFLCY(A,A,0,ADD);
1356   5                DUMMY = A + A;
1357   5                S = SIGN;
1358   5                DUMMY = A + A;
1359   5                Z = ZERO;
1360   5                A = A + A;
1361   5                P = PARITY;
1362   5                END;

                  /* 88: ADC B */
1363   4                DO;
1364   5                CALL OFLCY(A,B,CY,ADD);
1365   5                DUMMY = A + B + (CY AND 01H);
1366   5                S = SIGN;
1367   5                DUMMY = A + B + (CY AND 01H);
1368   5                Z = ZERO;
1369   5                A = A + B + (CY AND 01H);
1370   5                P = PARITY;
1371   5                END;
```

```
                /* 89: ADC C */
1372   4                DO;
1373   5                CALL OFLCY(A,C,CY,ADD);
1374   5                DUMMY = A + C + (CY AND 01H
1375   5                S = SIGN;
1376   5                DUMMY = A + C + (CY AND 01H)
1377   5                Z = ZERO;
1378   5                A = A + C + (CY AND 01H);
1379   5                P = PARITY;
1380   5                END;

                /* 8A: ADC D */
1381   4                DO;
1382   5                CALL OFLCY(A,D,CY,ADD);
1383   5                DUMMY = A + D + (CY AND 01H);
1384   5                S = SIGN;
1385   5                DUMMY = A + D + (CY AND 01H);
1386   5                Z = ZERO;
1387   5                A = A + D + (CY AND 01H);
1388   5                P = PARITY;
1389   5                END;

                /* 8B: ADC E */
1390   4                DO;
1391   5                CALL OFLCY(A,E,CY,ADD);
1392   5                DUMMY = A + E + (CY AND 01H);
1393   5                S = SIGN;
1394   5                DUMMY = A + E + (CY AND 01H);
1395   5                Z = ZERO;
1396   5                A = A + E + (CY AND 01H);
1397   5                P = PARITY;
1398   5                END;

                /* 8C: ADC H */
1399   4                DO;
1400   5                CALL OFLCY(A,H,CY,ADD);
1401   5                DUMMY = A + H + (CY AND 01H);
1402   5                S = SIGN;
1403   5                DUMMY = A + H + (CY AND 01H);
1404   5                Z = ZERO;
1405   5                A = A + H + (CY AND 01H);
1406   5                P = PARITY;
1407   5                END;

                /* 8D: ADC L */
1408   4                DO;
1409   5                CALL OFLCY(A,L,CY,ADD);
1410   5                DUMMY = A + L + (CY AND 01H);
1411   5                S = SIGN;
1412   5                DUMMY = A + L + (CY AND 01H);
1413   5                Z = ZERO;
1414   5                A = A + L + (CY AND 01H);
1415   5                P = PARITY;
1416   5                END;

                /* 8E: ADC M */
1417   4                DO;
```

```
1418   5              CALL OFLCY(A,MDATA,CY,ADD);
1419   5              DUMMY = A + MDATA + (CY AND 01H);
1420   5              S = SIGN;
1421   5              DUMMY = A + MDATA + (CY AND 01H);
1422   5              Z = ZERO;
1423   5              A = A + MDATA + (CY AND 01H);
1424   5              P = PARITY;
1425   5              END;

                    /* 8F: ADC A */
1426   4              DO;
1427   5              CALL OFLCY(A,A,CY,ADD);
1428   5              DUMMY = A + A + (CY AND 01H);
1429   5              S = SIGN;
1430   5              DUMMY = A + A + (CY AND 01H);
1431   5              Z = ZERO;
1432   5              A = A + A + (CY AND 01H);
1433   5              P = PARITY;
1434   5              END;

                    /* 90: SUB B */
1435   4              DO;
1436   5              CALL OFLCY(A,B,0,SUB);
1437   5              DUMMY = A - B;
1438   5              S = SIGN;
1439   5              DUMMY = A - B;
1440   5              Z = ZERO;
1441   5              A = A - B;
1442   5              P = PARITY;
1443   5              END;

                    /* 91: SUB C */
1444   4              DO;
1445   5              CALL OFLCY(A,C,0,SUB);
1446   5              DUMMY = A - C;
1447   5              S = SIGN;
1448   5              DUMMY = A - C;
1449   5              Z = ZERO;
1450   5              A = A - C;
1451   5              P = PARITY;
1452   5              END;

                    /* 92: SUB D */
1453   4              DO;
1454   5              CALL OFLCY(A,D,0,SUB);
1455   5              DUMMY = A - D;
1456   5              S = SIGN;
1457   5              DUMMY = A - D;
1458   5              Z = ZERO;
1459   5              A = A - D;
1460   5              P = PARITY;
1461   5              END;

                    /* 93: SUB E */
1462   4              DO;
1463   5              CALL OFLCY(A,E,0,SUB);
1464   5              DUMMY = A - E;
```

```
1465   5              S = SIGN;
1466   5              DUMMY = A - E;
1467   5              Z = ZERO;
1468   5              A = A - E;
1469   5              P = PARITY;
1470   5              END;

                    /* 94: SUB H */
1471   4              DO;
1472   5              CALL OFLCY(A, H, 0, SUB);
1473   5              DUMMY = A - H;
1474   5              S = SIGN;
1475   5              DUMMY = A - H;
1476   5              Z = ZERO;
1477   5              A = A - H;
1478   5              P = PARITY;
1479   5              END;

                    /* 95: SUB L */
1480   4              DO;
1481   5              CALL OFLCY(A, L, 0, SUB);
1482   5              DUMMY = A - L;
1483   5              S = SIGN;
1484   5              DUMMY = A - L;
1485   5              Z = ZERO;
1486   5              A = A - L;
1487   5              P = PARITY;
1488   5              END;

                    /* 96: SUB M */
1489   4              DO;
1490   5              CALL OFLCY(A, MDATA, 0, SUB);
1491   5              DUMMY = A - MDATA;
1492   5              S = SIGN;
1493   5              DUMMY = A - MDATA;
1494   5              Z = ZERO;
1495   5              A = A - MDATA;
1496   5              P = PARITY;
1497   5              END;

                    /* 97: SUB A */
1498   4              DO;
1499   5              CALL OFLCY(A, A, 0, SUB);
1500   5              DUMMY = A - A;
1501   5              S = SIGN;
1502   5              DUMMY = A - A;
1503   5              Z = ZERO;
1504   5              A = A - A;
1505   5              P = PARITY;
1506   5              END;

                    /* 98: SBB B */
1507   4              DO;
1508   5              CALL OFLCY(A, B, CY, SUB);
1509   5              DUMMY = A - B - (CY AND 01H);
1510   5              S = SIGN;
1511   5              DUMMY = A - B - (CY AND 01H);
```

```
1512   5              Z = ZERO;
1513   5              A = A - B - (CY AND 01H);
1514   5              P = PARITY;
1515   5              END;

                    /* 99: SBB C */
1516   4              DO;
1517   5              CALL OFLCY(A, C, CY, SUB);
1518   5              DUMMY = A - C - (CY AND 01H);
1519   5              S = SIGN;
1520   5              DUMMY = A - C - (CY AND 01H);
1521   5              Z = ZERO;
1522   5              A = A - C - (CY AND 01H);
1523   5              P = PARITY;
1524   5              END;

                    /* 9A: SBB D */
1525   4              DO;
1526   5              CALL OFLCY(A, D, CY, SUB);
1527   5              DUMMY = A - D - (CY AND 01H);
1528   5              S = SIGN;
1529   5              DUMMY = A - D - (CY AND 01H);
1530   5              Z = ZERO;
1531   5              A = A - D - (CY AND 01H);
1532   5              P = PARITY;
1533   5              END;

                    /* 9B: SBB E */
1534   4              DO;
1535   5              CALL OFLCY(A, E, CY, SUB);
1536   5              DUMMY = A - E - (CY AND 01H);
1537   5              S = SIGN;
1538   5              DUMMY = A - E - (CY AND 01H);
1539   5              Z = ZERO;
1540   5              A = A - E - (CY AND 01H);
1541   5              P = PARITY;
1542   5              END;

                    /* 9C: SBB H */
1543   4              DO;
1544   5              CALL OFLCY(A, H, CY, SUB);
1545   5              DUMMY = A - H - (CY AND 01H);
1546   5              S = SIGN;
1547   5              DUMMY = A - H - (CY AND 01H);
1548   5              Z = ZERO;
1549   5              A = A - H - (CY AND 01H);
1550   5              P = PARITY;
1551   5              END;

                    /* 9D: SBB L */
1552   4              DO;
1553   5              CALL OFLCY(A, L, CY, SUB);
1554   5              DUMMY = A - L - (CY AND 01H);
1555   5              S = SIGN;
1556   5              DUMMY = A - L - (CY AND 01H);
1557   5              Z = ZERO;
1558   5              A = A - L - (CY AND 01H);
```

```
1559   5              P = PARITY;
1560   5              END;

                   /* 9E: SBB M */
1561   4              DO;
1562   5              CALL OFLCY(A,MDATA,CY,SUB);
1563   5              DUMMY = A - MDATA - (CY AND 01H);
1564   5              S = SIGN;
1565   5              DUMMY = A - MDATA - (CY AND 01H);
1566   5              Z = ZERO;
1567   5              A = A - MDATA - (CY AND 01H);
1568   5              P = PARITY;
1569   5              END;

                   /* 9F: SBB A */
1570   4              DO;
1571   5              CALL OFLCY(A,A,CY,SUB);
1572   5              DUMMY = A - A - (CY AND 01H);
1573   5              S = SIGN;
1574   5              DUMMY = A - A - (CY AND 01H);
1575   5              Z = ZERO;
1576   5              A = A - A - (CY AND 01H);
1577   5              P = PARITY;
1578   5              END;

                   /* A0: ANA B */
1579   4              DO;
1580   5              DUMMY = A AND B;
1581   5              S = SIGN;
1582   5              DUMMY = A AND B;
1583   5              Z = ZERO;
1584   5              A = A AND B;
1585   5              P = PARITY;
1586   5              CY = 0;
1587   5              END;

                   /* A1: ANA C */
1588   4              DO;
1589   5              DUMMY = A AND C;
1590   5              S = SIGN;
1591   5              DUMMY = A AND C;
1592   5              Z = ZERO;
1593   5              A = A AND C;
1594   5              P = PARITY;
1595   5              CY = 0;
1596   5              END;

                   /* A2: ANA D */
1597   4              DO;
1598   5              DUMMY = A AND D;
1599   5              S = SIGN;
1600   5              DUMMY = A AND D;
1601   5              Z = ZERO;
1602   5              A = A AND D;
1603   5              P = PARITY;
1604   5              CY = 0;
1605   5              END;
```

```
                /* A3: ANA E */
1606    4               DO;
1607    5               DUMMY = A AND E;
1608    5               S = SIGN;
1609    5               DUMMY = A AND E;
1610    5               Z = ZERO;
1611    5               A = A AND E;
1612    5               P = PARITY;
1613    5               CY = 0;
1614    5               END;

                /* A4: ANA H */
1615    4               DO;
1616    5               DUMMY = A AND H;
1617    5               S = SIGN;
1618    5               DUMMY = A AND H;
1619    5               Z = ZERO;
1620    5               A = A AND H;
1621    5               P = PARITY;
1622    5               CY = 0;
1623    5               END;

                /* A5: ANA L */
1624    4               DO;
1625    5               DUMMY = A AND L;
1626    5               S = SIGN;
1627    5               DUMMY = A AND L;
1628    5               Z = ZERO;
1629    5               A = A AND L;
1630    5               P = PARITY;
1631    5               CY = 0;
1632    5               END;

                /* A6: ANA M */
1633    4               DO;
1634    5               DUMMY = A AND MDATA;
1635    5               S = SIGN;
1636    5               DUMMY = A AND MDATA;
1637    5               Z = ZERO;
1638    5               A = A AND MDATA;
1639    5               P = PARITY;
1640    5               CY = 0;
1641    5               END;

                /* A7: ANA A */
1642    4               DO;
1643    5               DUMMY = A AND A;
1644    5               S = SIGN;
1645    5               DUMMY = A AND A;
1646    5               Z = ZERO;
1647    5               A = A AND A;
1648    5               P = PARITY;
1649    5               CY = 0;
1650    5               END;

                /* A8: XRA B */
```

```
1651   4                DO;
1652   5                DUMMY = A XOR B;
1653   5                S = SIGN;
1654   5                DUMMY = A XOR B;
1655   5                Z = ZERO;
1656   5                A = A XOR B;
1657   5                P = PARITY;
1658   5                CY = 0;
1659   5                END;

                  /* A9: XRA C */
1660   4                DO;
1661   5                DUMMY = A XOR C;
1662   5                S = SIGN;
1663   5                DUMMY = A XOR C;
1664   5                Z = ZERO;
1665   5                A = A XOR C;
1666   5                P = PARITY;
1667   5                CY = 0;
1668   5                END;

                  /* AA: XRA D */
1669   4                DO;
1670   5                DUMMY = A XOR D;
1671   5                S = SIGN;
1672   5                DUMMY = A XOR D;
1673   5                Z = ZERO;
1674   5                A = A XOR D;
1675   5                P = PARITY;
1676   5                CY = 0;
1677   5                END;

                  /* AB: XRA E */
1678   4                DO;
1679   5                DUMMY = A XOR E;
1680   5                S = SIGN;
1681   5                DUMMY = A XOR E;
1682   5                Z = ZERO;
1683   5                A = A XOR E;
1684   5                P = PARITY;
1685   5                CY = 0;
1686   5                END;

                  /* AC: XRA H */
1687   4                DO;
1688   5                DUMMY = A XOR H;
1689   5                S = SIGN;
1690   5                DUMMY = A XOR H;
1691   5                Z = ZERO;
1692   5                A = A XOR H;
1693   5                P = PARITY;
1694   5                CY = 0;
1695   5                END;

                  /* AD: XRA L */
1696   4                DO;
1697   5                DUMMY = A XOR L;
```

```
1698   5                S = SIGN;
1699   5                DUMMY = A XOR L;
1700   5                Z = ZERO;
1701   5                A = A XOR L;
1702   5                P = PARITY;
1703   5                CY = 0;
1704   5                END;

                  /* AE: XRA M */
1705   4                DO;
1706   5                DUMMY = A XOR MDATA;
1707   5                S = SIGN;
1708   5                DUMMY = A XOR MDATA;
1709   5                Z = ZERO;
1710   5                A = A XOR MDATA;
1711   5                P = PARITY;
1712   5                CY = 0;
1713   5                END;

                  /* AF: XRA A */
1714   4                DO;
1715   5                DUMMY = A XOR A;
1716   5                S = SIGN;
1717   5                DUMMY = A XOR A;
1718   5                Z = ZERO;
1719   5                A = A XOR A;
1720   5                P = PARITY;
1721   5                CY = 0;
1722   5                END;

                  /* B0: ORA B */
1723   4                DO;
1724   5                DUMMY = A OR B;
1725   5                S = SIGN;
1726   5                DUMMY = A OR B;
1727   5                Z = ZERO;
1728   5                A = A OR B;
1729   5                P = PARITY;
1730   5                CY = 0;
1731   5                END;

                  /* B1: ORA C */
1732   4                DO;
1733   5                DUMMY = A OR C;
1734   5                S = SIGN;
1735   5                DUMMY = A OR C;
1736   5                Z = ZERO;
1737   5                A = A OR C;
1738   5                P = PARITY;
1739   5                CY = 0;
1740   5                END;

                  /* B2: ORA D */
1741   4                DO;
1742   5                DUMMY = A OR D;
1743   5                S = SIGN;
1744   5                DUMMY = A OR D;
```

```
1745   5                Z = ZERO;
1746   5                A = A OR D;
1747   5                P = PARITY;
1748   5                CY = 0;
1749   5                END;

                  /* B3: ORA E */
1750   4                DO;
1751   5                DUMMY = A OR E;
1752   5                S = SIGN;
1753   5                DUMMY = A OR E;
1754   5                Z = ZERO;
1755   5                A = A OR E;
1756   5                P = PARITY;
1757   5                CY = 0;
1758   5                END;

                  /* B4: ORA H */
1759   4                DO;
1760   5                DUMMY = A OR H;
1761   5                S = SIGN;
1762   5                DUMMY = A OR H;
1763   5                Z = ZERO;
1764   5                A = A OR H;
1765   5                P = PARITY;
1766   5                CY = 0;
1767   5                END;

                  /* B5: ORA L */
1768   4                DO;
1769   5                DUMMY = A OR L;
1770   5                S = SIGN;
1771   5                DUMMY = A OR L;
1772   5                Z = ZERO;
1773   5                A = A OR L;
1774   5                P = PARITY;
1775   5                CY = 0;
1776   5                END;

                  /* B6: ORA M */
1777   4                DO;
1778   5                DUMMY = A OR MDATA;
1779   5                S = SIGN;
1780   5                DUMMY = A OR MDATA;
1781   5                Z = ZERO;
1782   5                A = A OR MDATA;
1783   5                P = PARITY;
1784   5                CY = 0;
1785   5                END;

                  /* B7: ORA A */
1786   4                DO;
1787   5                DUMMY = A OR A;
1788   5                S = SIGN;
1789   5                DUMMY = A OR A;
1790   5                Z = ZERO;
1791   5                A = A OR A;
```

```
1792   5                P = PARITY;
1793   5                CY = 0;
1794   5                END;

                  /* B8: CMP B */
1795   4                DO;
1796   5                CALL OFLCY(A,B,0,SUB);
1797   5                DUMMY = A - B;
1798   5                S = SIGN;
1799   5                DUMMY = A - B;
1800   5                Z = ZERO;
1801   5                DUMMY = A - B;
1802   5                P = PARITY;
1803   5                END;

                  /* B9: CMP C */
1804   4                DO;
1805   5                CALL OFLCY(A,C,0,SUB);
1806   5                DUMMY = A - C;
1807   5                S = SIGN;
1808   5                DUMMY = A - C;
1809   5                Z = ZERO;
1810   5                DUMMY = A - C;
1811   5                P = PARITY;
1812   5                END;

                  /* BA: CMP D */
1813   4                DO;
1814   5                CALL OFLCY(A,D,0,SUB);
1815   5                DUMMY = A - D;
1816   5                S = SIGN;
1817   5                DUMMY = A - D;
1818   5                Z = ZERO;
1819   5                DUMMY = A - D;
1820   5                P = PARITY;
1821   5                END;

                  /* BB: CMP E */
1822   4                DO;
1823   5                CALL OFLCY(A,E,0,SUB);
1824   5                DUMMY = A - E;
1825   5                S = SIGN;
1826   5                DUMMY = A - E;
1827   5                Z = ZERO;
1828   5                DUMMY = A - E;
1829   5                P = PARITY;
1830   5                END;

                  /* BC: CMP H */
1831   4                DO;
1832   5                CALL OFLCY(A,H,0,SUB);
1833   5                DUMMY = A - H;
1834   5                S = SIGN;
1835   5                DUMMY = A - H;
1836   5                Z = ZERO;
1837   5                DUMMY = A - H;
1838   5                P = PARITY;
```

```
1839   5                END;

                  /* BD: CMP L */
1840   4                DO;
1841   5                CALL OFLCY(A,L,0,SUB);
1842   5                DUMMY = A - L;
1843   5                S = SIGN;
1844   5                DUMMY = A - L;
1845   5                Z = ZERO;
1846   5                DUMMY = A - L;
1847   5                P = PARITY;
1848   5                END;

                  /* BE: CMP M */
1849   4                DO;
1850   5                CALL OFLCY(A,MDATA,0,SUB);
1851   5                DUMMY = A - MDATA;
1852   5                S = SIGN;
1853   5                DUMMY = A - MDATA;
1854   5                Z = ZERO;
1855   5                DUMMY = A - MDATA;
1856   5                P = PARITY;
1857   5                END;

                  /* BF: CMP A */
1858   4                DO;
1859   5                CALL OFLCY(A,A,0,SUB);
1860   5                DUMMY = A - A;
1861   5                S = SIGN;
1862   5                DUMMY = A - A;
1863   5                Z = ZERO;
1864   5                DUMMY = A - A;
1865   5                P = PARITY;
1866   5                END;

                  /* C0: RNZ */
1867   4                DO;
1868   5                IF Z = 0 THEN PC = MDATA2;
1870   5                END;

                  /* C1: POP B */
1871   4                BC = MDATA2;

                  /* C2: JNZ ADDR */
1872   4                DO;
1873   5                IF Z = 0 THEN PC = ADDR;
1875   5                END;

                  /* C3: JMP ADDR */
1876   4                PC = ADDR;

                  /* C4: CNZ ADDR */
1877   4                DO;
1878   5                IF Z = 0 THEN DO;
1880   6                                SDATA2 = PC;
1881   6                                PC = ADDR;
1882   6                                END;
```

```
1883  5                       ELSE DO;
1884  6                            IF HLT AND HOLD THEN GOTO HHOLD;
1886  6                            IF HLT          THEN GOTO HALTE;
1888  6                            IF HOLD         THEN GOTO H0LD;
1890  6                            IF INT AND INTE THEN GOTO INTJAM1;
1892  6                                            ELSE GOTO IFCH1;
1893  6                            END;
1894  5               END;

                /* C5: PUSH B */
1895  4               SDATA2 = BC;

                /* C6: ADI DATA8 */
1896  4               DO;
1897  5               CALL OFLCY(A, DATABYTE, 0, ADD);
1898  5               DUMMY = A + DATABYTE;
1899  5               S = SIGN;
1900  5               DUMMY = A + DATABYTE;
1901  5               Z = ZERO;
1902  5               A = A + DATABYTE;
1903  5               P = PARITY;
1904  5               END;

                /* C7: RST 0 */
1905  4               DO;
1906  5               SDATA2 = PC;
1907  5               PC = 0;
1908  5               END;

                /* C8: RZ */
1909  4               DO;
1910  5               IF Z <> 0 THEN PC = MDATA2;
1912  5               END;

                /* C9: RET */
1913  4               PC = MDATA2;

                /* CA: JZ ADDR */
1914  4               DO;
1915  5               IF Z <> 0 THEN PC = ADDR;
1917  5               END;

                /* CB: HALT */
1918  4               DO;
1919  5               INTE = 0FFH;
1920  5               IF HLT AND HOLD THEN GOTO IEHHOLD;
1922  5               IF HLT          THEN GOTO HALTIE;
1924  5               IF HOLD         THEN GOTO IHHOLD;
1926  5               IF INT AND INTE THEN GOTO INTJAM1;
1928  5                                    ELSE GOTO HALTI;
1929  5               END;

                /* CC: CZ ADDR */
1930  4               DO;
1931  5               IF Z <> 0 THEN DO;
1933  6                                SDATA2 = PC;
1934  6                                PC = ADDR;
```

```
1935   6                              END;
1936   5                         ELSE DO;
1937   6                              IF HLT AND HOLD THEN GOTO HHOLD;
1939   6                              IF HLT          THEN GOTO HALTE;
1941   6                              IF HOLD         THEN GOTO HOLD;
1943   6                              IF INT AND INTE THEN GOTO INTJAM1;
1945   6                                              ELSE GOTO IFCH1;
1946   6                              END;
1947   5                END;

                  /* CD: CALL ADDR */
1948   4                DO;
1949   5                SDATA2 = PC;
1950   5                PC = ADDR;
1951   5                END;

                  /* CE: ACI DATA8 */
1952   4                DO;
1953   5                CALL OFLCY(A,DATABYTE,CY,ADD);
1954   5                DUMMY = A + DATABYTE + (CY AND 01H);
1955   5                S = SIGN;
1956   5                DUMMY = A + DATABYTE + (CY AND 01H);
1957   5                Z = ZERO;
1958   5                A = A + DATABYTE + (CY AND 01H);
1959   5                P = PARITY;
1960   5                END;

                  /* CF: RST 1 */
1961   4                DO;
1962   5                SDATA2 = PC;
1963   5                PC = 8;
1964   5                END;

                  /* D0: RNC */
1965   4                DO;
1966   5                IF CY = 0 THEN PC = MDATA2;
1968   5                END;

                  /* D1: POP D */
1969   4                DE = MDATA2;

                  /* D2: JNC ADDR */
1970   4                DO;
1971   5                IF CY = 0 THEN PC = ADDR;
1973   5                END;

                  /* D3: OUT PORT */
1974   4                SDATA = A;

                  /* D4: CNC ADDR */
1975   4                DO;
1976   5                IF CY = 0 THEN DO;
1978   6                                SDATA2 = PC;
1979   6                                PC = ADDR;
1980   6                                END;
1981   5                           ELSE DO;
1982   6                                IF HLT AND HOLD THEN GOTO HHOLD;
```

```
1984  6                          IF HLT          THEN GOTO HALTE;
1986  6                          IF HOLD         THEN GOTO HØLD;
1988  6                          IF INT AND INTE THEN GOTO INTJAM1;
1990  6                                          ELSE GOTO IFCH1;
1991  6                          END;
1992  5            END;

                /* D5: PUSH D */
1993  4            SDATA2 = DE;

                /* D6: SUI DATA8 */
1994  4            DO;
1995  5            CALL OFLCY(A, DATABYTE, 0, SUB);
1996  5            DUMMY = A - DATABYTE;
1997  5            S = SIGN;
1998  5            DUMMY = A - DATABYTE;
1999  5            Z = ZERO;
2000  5            A = A - DATABYTE;
2001  5            P = PARITY;
2002  5            END;

                /* D7: RST 2 */
2003  4            DO;
2004  5            SDATA2 = PC;
2005  5            PC = 16;
2006  5            END;

                /* D8: RC */
2007  4            DO;
2008  5            IF CY <> 0 THEN PC = MDATA2;
2010  5            END;

                /* D9: HALT */
2011  4            DO;
2012  5            INTE = 0FFH;
2013  5            IF HLT AND HOLD THEN GOTO IEHHOLD;
2015  5            IF HLT          THEN GOTO HALTIE;
2017  5            IF HOLD         THEN GOTO IHHOLD;
2019  5            IF INT AND INTE THEN GOTO INTJAM1;
2021  5                                 ELSE GOTO HALTI;
2022  5            END;

                /* DA: JC ADDR */
2023  4            DO;
2024  5            IF CY <> 0 THEN PC = ADDR;
2026  5            END;

                /* DB: IN PORT */
2027  4            A = MDATA;

                /* DC: CC ADDR */
2028  4            DO;
2029  5            IF CY <> 0 THEN DO;
2031  6                                 SDATA2 = PC;
2032  6                                 PC = ADDR;
2033  6                                 END;
2034  5                            ELSE DO;
```

```
2035   6                          IF HLT AND HOLD THEN GOTO HHOLD;
2037   6                          IF HLT          THEN GOTO HALTE;
2039   6                          IF HOLD         THEN GOTO HOLD;
2041   6                          IF INT AND INTE THEN GOTO INTJAM1;
2043   6                                          ELSE GOTO IFCH1;
2044   6                          END;
2045   5              END;

                    /* DD: HALT */
2046   4              DO;
2047   5              INTE = 0FFH;
2048   5              IF HLT AND HOLD THEN GOTO IEHHOLD;
2050   5              IF HLT          THEN GOTO HALTIE;
2052   5              IF HOLD         THEN GOTO IHHOLD;
2054   5              IF INT AND INTE THEN GOTO INTJAM1;
2056   5                              ELSE GOTO HALTI;
2057   5              END;

                    /* DE: SBI DATA8 */
2058   4              DO;
2059   5              CALL OFLCY(A, DATABYTE, CY, SUB);
2060   5              DUMMY = A - DATABYTE - (CY AND 01H);
2061   5              S = SIGN;
2062   5              DUMMY = A - DATABYTE - (CY AND 01H);
2063   5              Z = ZERO;
2064   5              A = A - DATABYTE - (CY AND 01H);
2065   5              P = PARITY;
2066   5              END;

                    /* DF: RST 3 */
2067   4              DO;
2068   5              SDATA2 = PC;
2069   5              PC = 24;
2070   5              END;

                    /* E0: RPO */
2071   4              DO;
2072   5              IF P = 0 THEN PC = MDATA2;
2074   5              END;

                    /* E1: POP H */
2075   4              HL = MDATA2;

                    /* E2: JPO ADDR */
2076   4              DO;
2077   5              IF P = 0 THEN PC = ADDR;
2079   5              END;

                    /* E3: XTHL */
2080   4              DO;
2081   5              SDATA2 = HL;
2082   5              HL = MDATA2;
2083   5              END;

                    /* E4: CPO ADDR */
2084   4              DO;
2085   5              IF P = 0 THEN DO;
```

```
2087   6                              SDATA2 = PC;
2088   6                              PC = ADDR;
2089   6                              END;
2090   5                         ELSE DO;
2091   6                              IF HLT AND HOLD THEN GOTO HHOLD;
2093   6                              IF HLT          THEN GOTO HALTE;
2095   6                              IF HOLD         THEN GOTO HOLD;
2097   6                              IF INT AND INTE THEN GOTO INTJAM1;
2099   6                                              ELSE GOTO IFCH1;
2100   6                              END;
2101   5                    END;

                      /* E5: PUSH H */
2102   4                    SDATA2 = HL;

                      /* E6: ANI DATA8 */
2103   4                    DO;
2104   5                    DUMMY = A AND DATABYTE;
2105   5                    S = SIGN;
2106   5                    DUMMY = A AND DATABYTE;
2107   5                    Z = ZERO;
2108   5                    A = A AND DATABYTE;
2109   5                    P = PARITY;
2110   5                    CY = 0;
2111   5                    END;

                      /* E7: RST 4 */
2112   4                    DO;
2113   5                    SDATA2 = PC;
2114   5                    PC = 32;
2115   5                    END;

                      /* E8: RPE */
2116   4                    DO;
2117   5                    IF P <> 0 THEN PC = MDATA2;
2119   5                    END;

                      /* E9: PCHL */
2120   4                    PC = HL;

                      /* EA: JPE ADDR */
2121   4                    DO;
2122   5                    IF P <> 0 THEN PC = ADDR;
2124   5                    END;

                      /* EB: XCHG */
2125   4                    DO;
2126   5                    MDATA2 = DE;
2127   5                    DE = HL;
2128   5                    HL = MDATA2;
2129   5                    END;

                      /* EC: CPE ADDR */
2130   4                    DO;
2131   5                    IF P <> 0 THEN DO;
2133   6                                   SDATA2 = PC;
2134   6                                   PC = ADDR;
```

```
2135  6                          END;
2136  5                     ELSE DO;
2137  6                          IF HLT AND HOLD THEN GOTO HHOLD;
2139  6                          IF HLT          THEN GOTO HALTE;
2141  6                          IF HOLD         THEN GOTO H0LD;
2143  6                          IF INT AND INTE THEN GOTO INTJAM1;
2145  6                                               ELSE GOTO IFCH1;
2146  6                          END;
2147  5               END;

                    /* ED: HALT */
2148  4               DO;
2149  5               INTE = 0FFH;
2150  5               IF HLT AND HOLD THEN GOTO IEHHOLD;
2152  5               IF HLT          THEN GOTO HALTIE;
2154  5               IF HOLD         THEN GOTO IHHOLD;
2156  5               IF INT AND INTE THEN GOTO INTJAM1;
2158  5                                    ELSE GOTO HALTI;
2159  5               END;

                    /* EE: XRI DATA8 */
2160  4               DO;
2161  5               DUMMY = A XOR DATABYTE;
2162  5               S = SIGN;
2163  5               DUMMY = A XOR DATABYTE;
2164  5               Z = ZERO;
2165  5               A = A XOR DATABYTE;
2166  5               P = PARITY;
2167  5               CY = 0;
2168  5               END;

                    /* EF: RST 5 */
2169  4               DO;
2170  5               SDATA2 = PC;
2171  5               PC = 40;
2172  5               END;

                    /* F0: RP */
2173  4               DO;
2174  5               IF S = 0 THEN PC = MDATA2;
2176  5               END;

                    /* F1: POP PSW */
2177  4               DO;
2178  5               A     = HIGH(MDATA2);
2179  5               DUMMY = LOW (MDATA2);
2180  5               IF     DUMMY    THEN CY = 0FFH;
2182  5                               ELSE CY = 0;
2183  5               IF SHR(DUMMY,2) THEN P  = 0FFH;
2185  5                               ELSE P  = 0;
2186  5               IF SHR(DUMMY,6) THEN Z  = 0FFH;
2188  5                               ELSE Z  = 0;
2189  5               IF SHR(DUMMY,7) THEN S  = 0FFH;
2191  5                               ELSE S  = 0;
2192  5               END;

                    /* F2: JP ADDR */
```

```
2193  4                 DO;
2194  5                 IF S = 0 THEN PC = ADDR;
2196  5                 END;

                  /* F3: DI */
2197  4                 INTE = 0;

                  /* F4: CP ADDR */
2198  4                 DO;
2199  5                 IF S = 0 THEN DO;
2201  6                                 SDATA2 = PC;
2202  6                                 PC = ADDR;
2203  6                                 END;
2204  5                           ELSE DO;
2205  6                                 IF HLT AND HOLD THEN GOTO HHOLD;
2207  6                                 IF HLT          THEN GOTO HALTE;
2209  6                                 IF HOLD         THEN GOTO H0LD;
2211  6                                 IF INT AND INTE THEN GOTO INTJAM1;
2213  6                                                 ELSE GOTO IFCH1;
2214  6                                 END;
2215  5                 END;

                  /* F5: PUSH PSW */
2216  4                 DO;
2217  5                 SDATAH = A;
2218  5                 SDATA  = (CY AND 01H) OR
                                   SHL(P AND 01H,2) OR
                                   SHL(Z AND 01H,6) OR
                                   SHL(S AND 01H,7) OR
                                   00000010B;
2219  5                 END;

                  /* F6: ORI DATA8 */
2220  4                 DO;
2221  5                 DUMMY = A OR DATABYTE;
2222  5                 S = SIGN;
2223  5                 DUMMY = A OR DATABYTE;
2224  5                 Z = ZERO;
2225  5                 A = A OR DATABYTE;
2226  5                 P = PARITY;
2227  5                 CY = 0;
2228  5                 END;

                  /* F7: RST 6 */
2229  4                 DO;
2230  5                 SDATA2 = PC;
2231  5                 PC = 48;
2232  5                 END;

                  /* F8: RM */
2233  4                 DO;
2234  5                 IF S <> 0 THEN PC = MDATA2;
2236  5                 END;

                  /* F9: SPHL */
2237  4                 SP = HL;
```

```
                /* FA: JM ADDR */
2238   4                DO;
2239   5                IF S <> 0 THEN PC = ADDR;
2241   5                END;

                /* FB: EI */
2242   4                INTE = 0FFH;

                /* FC: CM ADDR */
2243   4                DO;
2244   5                IF S <> 0 THEN DO;
2246   6                                SDATA2 = PC;
2247   6                                PC = ADDR;
2248   6                                END;
2249   5                           ELSE DO;
2250   6                                IF HLT AND HOLD THEN GOTO HHOLD;
2252   6                                IF HLT          THEN GOTO HALTE;
2254   6                                IF HOLD         THEN GOTO H0LD;
2256   6                                IF INT AND INTE THEN GOTO INTJAM1;
2258   6                                                ELSE GOTO IFCH1;
2259   6                                END;
2260   5                END;

                /* FD: HALT */
2261   4                DO;
2262   5                INTE = 0FFH;
2263   5                IF HLT AND HOLD THEN GOTO IEHHOLD;
2265   5                IF HLT          THEN GOTO HALTIE;
2267   5                IF HOLD         THEN GOTO IHHOLD;
2269   5                IF INT AND INTE THEN GOTO INTJAM1;
2271   5                                ELSE GOTO HALTI;
2272   5                END;

                /* FE: CPI DATA8 */
2273   4                DO;
2274   5                CALL OFLCY(A,DATABYTE,0,SUB);
2275   5                DUMMY = A - DATABYTE;
2276   5                S = SIGN;
2277   5                DUMMY = A - DATABYTE;
2278   5                Z = ZERO;
2279   5                DUMMY = A - DATABYTE;
2280   5                P = PARITY;
2281   5                END;

                /* FF: RST 7 */
2282   4                DO;
2283   5                SDATA2 = PC;
2284   5                PC = 56;
2285   5                END;

2286   4                END;    /* OF DO CASE */

2287   3                DO CASE SHR(TBL(OPCODE) AND 06H,1);
2288   4                       DO;                          /* ST = 00 */
2289   5                       IF HLT AND HOLD THEN GOTO HHOLD;
2291   5                       IF HLT          THEN GOTO HALTE;
2293   5                       IF HOLD         THEN GOTO H0LD;
```

```
2295   5                    IF INT AND INTE THEN GOTO INTJAM1;
2297   5                                    ELSE GOTO IFCH1;
2298   5                    END;
2299   4                    GOTO STORE1;            /* ST = 01 */
2300   4                    GOTO PUSH1;             /* ST = 10 */
2301   4                    GOTO OUTPUT1;           /* ST = 11 */
2302   4              END;
2303   3              END;
```

```
                $EJECT
                                /*************/
                                /*  STORE1   */
                                /*************/

2304    2       STORE1: DO;
2305    3               IF OPCODE = 02H THEN CALL MOVE(2,.BC,.V);
2307    3               IF OPCODE = 12H THEN CALL MOVE(2,.DE,.V);
2309    3               IF OPCODE = 22H THEN CALL MOVE(2,.ADDR,.V);
2311    3               IF OPCODE = 32H THEN CALL MOVE(2,.ADDR,.V);
2313    3               IF (OPCODE AND 0CFH) <> 02H THEN CALL MOVE(2,.HL,.V);
2315    3               V(2) = SDATA;
2316    3               CALL SETBIT(.V,26,1,INTE);
2317    3               CALL SETBIT(.V,27,1,0B);
2318    3               CALL SETBIT(.V,29,1,0B);
2319    3               CALL SETBIT(.V,31,1,0B);
2320    3               CALL SETBIT(.V,33,1,OFL);
2321    3               CALL SETBIT(.V,34,6,000001B);
2322    3               DEF = OLDDEF AND 0FFEFH OR 0003H;

2323    3               CALL ALPHA(80,6,.V);
2324    3               CALL SCHEDULE(8);
2325    3               CALL MEMW(MADDR,SDATA);
2326    3               CALL PRINT;

2327    3               IF RESIN THEN GOTO RESET;
2329    3               IF (TBL(OPCODE) AND 01H) = 01H THEN GOTO STORE2;
2331    3               IF HLT AND HOLD THEN GOTO HHOLD;
2333    3               IF HLT          THEN GOTO HALTE;
2335    3               IF HOLD         THEN GOTO H0LD;
2337    3               IF INT AND INTE THEN GOTO INTJAM1;
2339    3                                  ELSE GOTO IFCH1;
2340    3               END;
```

AD-A063 004 QUESTRON CORP SAN DIEGO CALIF F/G 9/2
RADIATION HARDENED MICROPROCESSOR. VOLUME II.(U)
MAY 78 V V NICKEL, P A ROSENBERG F33615-77-C-1001
UNCLASSIFIED AFAL-TR-78-55-VOL-2 NL

4 OF 4
AD
A063 004

END
DATE
FILMED
3-79
DDC

```
                $EJECT
                                /**************/
                                /*  STORE2    */
                                /**************/

2341   2        STORE2: DO;
2342   3                V(0) = LOW (MADDR+1);
2343   3                V(1) = HIGH(MADDR+1);
2344   3                V(2) = HIGH(SDATA2);
2345   3                CALL SETBIT(.V,26,1,INTE);
2346   3                CALL SETBIT(.V,27,1,0B);
2347   3                CALL SETBIT(.V,29,1,0B);
2348   3                CALL SETBIT(.V,31,1,0B);
2349   3                CALL SETBIT(.V,33,1,OFL);
2350   3                CALL SETBIT(.V,34,6,000001B);
2351   3                DEF = OLDDEF;

2352   3                CALL ALPHA(B0,6,.V);
2353   3                CALL SCHEDULE(9);
2354   3                CALL MEMW(MADDR+1,HIGH(SDATA2));
2355   3                CALL PRINT;

2356   3                IF RESIN         THEN GOTO RESET;
2358   3                IF HLT AND HOLD  THEN GOTO HHOLD;
2360   3                IF HLT           THEN GOTO HALTE;
2362   3                IF HOLD          THEN GOTO H0LD;
2364   3                IF INT AND INTE  THEN GOTO INTJAM1;
2366   3                                 ELSE GOTO IFCH1;
2367   3                END;
```

```
                $EJECT
                                /**************/
                                /*  PUSH1  */
                                /**************/

2368   2        PUSH1:  DO;
2369   3                SP = SP - 1;
2370   3                V(0) = LOW (SP);
2371   3                V(1) = HIGH(SP);
2372   3                V(2) = HIGH(SDATA2);
2373   3                CALL SETBIT(.V,26,1,INTE);
2374   3                CALL SETBIT(.V,27,1,0B);
2375   3                CALL SETBIT(.V,29,1,0B);
2376   3                CALL SETBIT(.V,31,1,0B);
2377   3                CALL SETBIT(.V,33,1,OFL);
2378   3                CALL SETBIT(.V,34,6,100010B);
2379   3                DEF = OLDDEF AND 0FFEFH OR 0003H;

2380   3                CALL ALPHA(80,6,.V);
2381   3                CALL SCHEDULE(10);
2382   3                CALL MEMW(SP,HIGH(SDATA2));
2383   3                CALL PRINT;

2384   3                IF RESIN THEN GOTO RESET;
2386   3                         ELSE GOTO PUSH2;
2387   3                END;
```

```
                $EJECT
                                /**************/
                                /*  PUSH2  */
                                /**************/

2388   2        PUSH2:  DO;
2389   3                SP = SP - 1;
2390   3                V(0) = LOW (SP);
2391   3                V(1) = HIGH(SP);
2392   3                V(2) = SDATA;
2393   3                CALL SETBIT(.V,26,1,INTE);
2394   3                CALL SETBIT(.V,27,1,0B);
2395   3                CALL SETBIT(.V,29,1,0B);
2396   3                CALL SETBIT(.V,31,1,0B);
2397   3                CALL SETBIT(.V,33,1,OFL);
2398   3                CALL SETBIT(.V,34,6,100010B);
2399   3                DEF = OLDDEF;

2400   3                CALL ALPHA(B0,6,.V);
2401   3                CALL SCHEDULE(11);
2402   3                CALL MEMW(SP,SDATA);
2403   3                CALL PRINT;

2404   3                IF RESIN         THEN GOTO RESET;
2406   3                IF HLT AND HOLD  THEN GOTO HHOLD;
2408   3                IF HLT           THEN GOTO HALTE;
2410   3                IF HOLD          THEN GOTO H0LD;
2412   3                IF INT AND INTE  THEN GOTO INTJAM1;
2414   3                                 ELSE GOTO IFCH1;
2415   3                END;
```

```
                $EJECT
                                /**************/
                                /*  OUTPUT    */
                                /**************/

2416   2        OUTPUT1:DO;
2417   3                V(0) = DATABYTE;
2418   3                V(1) = 0;
2419   3                V(2) = SDATA;
2420   3                CALL SETBIT(.V,26,1,INTE);
2421   3                CALL SETBIT(.V,27,1,0B);
2422   3                CALL SETBIT(.V,29,1,0B);
2423   3                CALL SETBIT(.V,31,1,0B);
2424   3                CALL SETBIT(.V,33,1,0B);
2425   3                CALL SETBIT(.V,34,6,000100B);
2426   3                DEF = OLDDEF AND 0FFEFH OR 0003H;

2427   3                CALL ALPHA(BO,6,.V);
2428   3                CALL SCHEDULE(12);
2429   3                CALL PRINT;

2430   3                IF RESIN         THEN GOTO RESET;
2432   3                IF HLT AND HOLD  THEN GOTO HHOLD;
2434   3                IF HLT           THEN GOTO HALTE;
2436   3                IF HOLD          THEN GOTO H0LD;
2438   3                IF INT AND INTE  THEN GOTO INTJAM1;
2440   3                                 ELSE GOTO IFCH1;
2441   3                END;
```

```
                $EJECT
                                /**************/
                                /*  HALTE  */
                                /**************/

2442   2        HALTE:  DO;
2443   3                V(2) = LASTMEMTOQ80;
2444   3                CALL SETBIT(.V,26,1,INTE);
2445   3                CALL SETBIT(.V,27,1,0B);
2446   3                CALL SETBIT(.V,29,1,0B);
2447   3                CALL SETBIT(.V,31,1,1B);
2448   3                CALL SETBIT(.V,33,1,OFL);
2449   3                CALL SETBIT(.V,34,6,000000B);
2450   3                DEF = OLDDEF AND 0FFECH;

2451   3                CALL ALPHA(80,6,.V);
2452   3                CALL SCHEDULE(17);
2453   3                CALL PRINT;

2454   3                IF RESIN         THEN GOTO RESET;
2456   3                IF HLT AND HOLD  THEN GOTO HHOLD;
2458   3                IF HLT           THEN GOTO HALTE;
2460   3                IF HOLD          THEN GOTO H0LD;
2462   3                IF INT AND INTE  THEN GOTO INTJAM1;
2464   3                                 ELSE GOTO IFCH1;
2465   3                END;
```

```
                $EJECT
                                /************/
                                /*  HOLD    */
                                /************/

2466   2        H0LD:   DO;
2467   3                DECLARE OLDADDR ADDRESS;
2468   3                CALL MOVE(2,.V,.OLDADDR);
2469   3                V(0) = 0FFH;
2470   3                V(1) = 0FFH;
2471   3                V(2) = 0FFH;
2472   3                CALL SETBIT(.V,26,1,INTE);
2473   3                CALL SETBIT(.V,27,1,0B);
2474   3                CALL SETBIT(.V,29,1,1B);
2475   3                CALL SETBIT(.V,31,1,0B);
2476   3                CALL SETBIT(.V,33,1,OFL);
2477   3                CALL SETBIT(.V,34,6,000000B);
2478   3                DEF = OLDDEF AND 0FFEFH OR 0003H;

2479   3                CALL ALPHA(B0,6,.V);
2480   3                CALL SCHEDULE(18);
2481   3                CALL PRINT;

2482   3                CALL MOVE(2,.OLDADDR,.V);
2483   3                V(2) = LASTMEMTOQ80;

2484   3                IF RESIN         THEN GOTO RESET;
2486   3                IF HLT AND HOLD  THEN GOTO HHOLD;
2488   3                IF HOLD          THEN GOTO H0LD;
2490   3                IF HLT           THEN GOTO HALTE;
2492   3                IF INT AND INTE  THEN GOTO INTJAM1;
2494   3                                 ELSE GOTO IFCH1;
2495   3                END;
```

```
                $EJECT
                                /************/
                                /*  HHOLD   */
                                /************/

2496   2        HHOLD:  DO;
2497   3                DECLARE OLDADDR ADDRESS;
2498   3                CALL MOVE(2,.V,.OLDADDR);
2499   3                V(0) = 0FFH;
2500   3                V(1) = 0FFH;
2501   3                V(2) = 0FFH;
2502   3                CALL SETBIT(.V,26,1,INTE);
2503   3                CALL SETBIT(.V,27,1,0B);
2504   3                CALL SETBIT(.V,29,1,1B);
2505   3                CALL SETBIT(.V,31,1,1B);
2506   3                CALL SETBIT(.V,33,1,OFL);
2507   3                CALL SETBIT(.V,34,6,000000B);
2508   3                DEF = OLDDEF AND 0FFEFH OR 0003H;

2509   3                CALL ALPHA(80,6,.V);
2510   3                CALL SCHEDULE(19);
2511   3                CALL PRINT;

2512   3                CALL MOVE(2,.OLDADDR,.V);
2513   3                V(2) = LASTMEMTOQ80;

2514   3                IF RESIN         THEN GOTO RESET;
2516   3                IF HLT AND HOLD  THEN GOTO HHOLD;
2518   3                IF HLT           THEN GOTO HALTE;
2520   3                IF HOLD          THEN GOTO HOLD;
2522   3                IF INT AND INTE  THEN GOTO INTJAM1;
2524   3                                 ELSE GOTO IFCH1;
2525   3                END;
```

```
                $EJECT
                                /************/
                                /*  HALTI   */
                                /************/

2526   2        HALTI:  DO;
2527   3                V(2) = LASTMEMTOQ80;
2528   3                CALL SETBIT(.V,26,1,INTE);
2529   3                CALL SETBIT(.V,27,1,0B);
2530   3                CALL SETBIT(.V,29,1,0B);
2531   3                CALL SETBIT(.V,31,1,1B);
2532   3                CALL SETBIT(.V,33,1,0B);
2533   3                CALL SETBIT(.V,34,6,000000B);
2534   3                DEF = OLDDEF AND 0FFECH;

2535   3                CALL ALPHA(80,6,.V);
2536   3                CALL SCHEDULE(20);
2537   3                CALL PRINT;

2538   3                IF RESIN THEN GOTO RESET;
2540   3                DO CASE HLT AND 02H OR HOLD AND 01H;
2541   4                        DO;
2542   5                        IF INT AND INTE                    /* HH = 00 */
                                    THEN GOTO INTJAM1;
2544   5                                ELSE DO;
2545   6                                     CALL PAGE(LP);
2546   6                                     CALL PAGE(LP);
2547   6                                     CALL EXIT;
2548   6                                     END;
2549   5                        END;
2550   4                        GOTO IHHOLD;                       /* HH = 01 */
2551   4                        GOTO HALTIE;                       /* HH = 10 */
2552   4                        GOTO IEHHOLD;                      /* HH = 11 */
2553   4                END;
2554   3                END;
```

```
                $EJECT
                                /**************/
                                /*  HALTIE    */
                                /**************/

2555    2       HALTIE: DO;
2556    3               V(2) = LASTMEMTOQ80;
2557    3               CALL SETBIT(.V,26,1,INTE);
2558    3               CALL SETBIT(.V,27,1,0B);
2559    3               CALL SETBIT(.V,29,1,0B);
2560    3               CALL SETBIT(.V,31,1,1B);
2561    3               CALL SETBIT(.V,33,1,0B);
2562    3               CALL SETBIT(.V,34,6,000000B);
2563    3               DEF = OLDDEF AND 0FFECH;

2564    3               CALL ALPHA(BO,6,.V);
2565    3               CALL SCHEDULE(21);
2566    3               CALL PRINT;

2567    3               IF RESIN THEN GOTO RESET;
2569    3               DO CASE HLT AND 02H OR HOLD AND 01H;
2570    4                       DO;
2571    5                       IF INT AND INTE      /* HH = 00 */
                                   THEN GOTO INTJAM1;
2573    5                          ELSE DO;
2574    6                               CALL PAGE(LP);
2575    6                               CALL PAGE(LP);
2576    6                               CALL EXIT;
2577    6                               END;
2578    5                       END;
2579    4                       GOTO IHHOLD;         /* HH = 01 */
2580    4                       GOTO HALTIE;         /* HH = 10 */
2581    4                       GOTO IEHHOLD;        /* HH = 11 */
2582    4               END;
2583    3               END;
```

```
                $EJECT
                                /**************/
                                /*  IHHOLD    */
                                /**************/

2584   2        IHHOLD: DO;
2585   3                DECLARE OLDADDR ADDRESS;
2586   3                CALL MOVE(2,.V,.OLDADDR);
2587   3                V(0) = 0FFH;
2588   3                V(1) = 0FFH;
2589   3                V(2) = 0FFH;
2590   3                CALL SETBIT(.V,26,1,INTE);
2591   3                CALL SETBIT(.V,27,1,0B);
2592   3                CALL SETBIT(.V,29,1,1B);
2593   3                CALL SETBIT(.V,31,1,1B);
2594   3                CALL SETBIT(.V,33,1,0B);
2595   3                CALL SETBIT(.V,34,6,000000B);
2596   3                DEF = OLDDEF AND 0FFEFH OR 0003H;

2597   3                CALL ALPHA(80,6,.V);
2598   3                CALL SCHEDULE(22);
2599   3                CALL PRINT;

2600   3                CALL MOVE(2,.OLDADDR,.V);
2601   3                V(2) = LASTMEMTOQ80;

2602   3                IF RESIN THEN GOTO RESET;
2604   3                DO CASE HLT AND 02H OR HOLD AND 01H;
2605   4                        DO;
2606   5                        IF INT AND INTE  /* HH = 00 */
                                   THEN GOTO INTJAM1;
2608   5                                ELSE DO;
2609   6                                     CALL PAGE(LP);
2610   6                                     CALL PAGE(LP);
2611   6                                     CALL EXIT;
2612   6                                     END;
2613   5                        END;
2614   4                        GOTO IHHOLD;      /* HH = 01 */
2615   4                        GOTO HALTIE;      /* HH = 10 */
2616   4                        GOTO IEHHOLD;     /* HH = 11 */
2617   4                END;
2618   3                END;
```

```
                $EJECT
                                /***************/
                                /*  IEHHOLD    */
                                /***************/

2619    2       IEHHOLD:DO;
2620    3               DECLARE OLDADDR ADDRESS;
2621    3               CALL MOVE(2,.V,.OLDADDR);
2622    3               CALL SETBIT(.V,26,1,INTE);
2623    3               CALL SETBIT(.V,27,1,0B);
2624    3               CALL SETBIT(.V,29,1,1B);
2625    3               CALL SETBIT(.V,31,1,1B);
2626    3               CALL SETBIT(.V,33,1,0B);
2627    3               CALL SETBIT(.V,34,6,000000B);
2628    3               DEF = OLDDEF AND 0FFEFH OR 0003H;

2629    3               CALL ALPHA(80,6,.V);
2630    3               CALL SCHEDULE(23);
2631    3               CALL PRINT;

2632    3               CALL MOVE(2,.OLDADDR,.V);
2633    3               V(2) = LASTMEMTOQ80;

2634    3               IF RESIN THEN GOTO RESET;
2636    3               DO CASE HLT AND 02H OR HOLD AND 01H;
2637    4                       DO;
2638    5                       IF INT AND INTE  /* HH = 00 */
                                   THEN GOTO INTJAM1;
2640    5                          ELSE DO;
2641    6                                CALL PAGE(LP);
2642    6                                CALL PAGE(LP);
2643    6                                CALL EXIT;
2644    6                                END;
2645    5                       END;
2646    4                       GOTO IHHOLD;      /* HH = 01 */
2647    4                       GOTO HALTIE;      /* HH = 10 */
2648    4                       GOTO IEHHOLD;     /* HH = 11 */
2649    4               END;
2650    3               END;
```

```
                $EJECT
                                /***************/
                                /*  INTJAM1    */
                                /***************/

2651   2        INTJAM1:DO;
2652   3                INTE = 0;
2653   3                V(0) = LOW (PC);
2654   3                V(1) = HIGH(PC);
2655   3                V(2) = LASTMEMTOQ80;
2656   3                CALL SETBIT(.V,26,1,INTE);
2657   3                CALL SETBIT(.V,27,1,1B);
2658   3                CALL SETBIT(.V,29,1,0B);
2659   3                CALL SETBIT(.V,31,1,0B);
2660   3                CALL SETBIT(.V,33,1,OFL);
2661   3                CALL SETBIT(.V,34,6,010000B);
2662   3                DEF = OLDDEF AND 0FFFDH OR 0011H;

2663   3                CALL ALPHA(80,6,.V);
2664   3                CALL SCHEDULE(13);
2665   3                OPCODE, LASTMEMTOQ80 = INTJAMWORD(0);
2666   3                CALL PRINT;

2667   3                OFL = 0;

2668   3                IF RESIN THEN GOTO RESET;
2670   3                        ELSE GOTO INTJAM2;
2671   3                END;
```

```
                $EJECT
                                /***************/
                                /*  INTJAM2    */
                                /***************/

2672    2       INTJAM2:DO;
2673    3               V(2) = OPCODE;
2674    3               CALL SETBIT(.V,33,1,OFL);
2675    3               DEF = OLDDEF;

2676    3               CALL ALPHA(80,6,.V);
2677    3               CALL SCHEDULE(14);
2678    3               DATABYTE, LASTMEMTOQ80 = INTJAMWORD(1);
2679    3               CALL PRINT;

2680    3               IF RESIN THEN GOTO RESET;
2682    3                       ELSE GOTO INTJAM3;
2683    3               END;
```

```
            $EJECT
                              /***************/
                              /*  INTJAM3    */
                              /***************/

2684   2    INTJAM3:DO;
2685   3            V(2) = DATABYTE;
2686   3            DEF = OLDDEF;

2687   3            CALL ALPHA(80,6,.V);
2688   3            CALL SCHEDULE(15);
2689   3            ADDRH, LASTMEMTOQ80 = INTJAMWORD(2);
2690   3            CALL PRINT;

2691   3            IF RESIN THEN GOTO RESET;
2693   3            DO CASE SHR(TBL(OPCODE) AND 30H,4);
2694   4                    GOTO EXEC;          /* OF = 00 */
2695   4                    GOTO OFCH1;         /* OF = 01 */
2696   4                    GOTO POP1;          /* OF = 10 */
2697   4                    GOTO INPUT;         /* OF = 11 */
2698   4            END;
2699   3            END;
2700   2            END;   /* OF Q-80 SIMULATION BLOCK */
2701   1            END Q80SIM;


MODULE INFORMATION:

     CODE AREA SIZE     = 5FABH  24491D
     VARIABLE AREA SIZE = 0763H   1891D
     MAXIMUM STACK SIZE = 000EH     14D
     3500 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

6. Conclusions

The basic conclusion of this emulation analysis effort is that the GPU and its supporting devices (GUA, ROM and RAM) constitute a viable microprocessor bit slice device family. However, deficiencies were found in the GPU and its family members that must be rectified to ensure a truely competitive product line capable of handling Air Force requirements. GPU design deficiencies are presently being corrected in a reworked design. More importantly, deficiencies in the breadth of the device family were also found to exist and recommendations for additional part types for the family are proposed. The most important additional device type required is a microprogram controller. ~~The recommendation is to build~~ a CMOS/SOS version of the AMD 2910 is recommended.

A parallel effort was undertaken to define and then design, to the gate level, a particular computer for emulation ~~utilizing~~ using the device family. This effort provided a test of the emulation capabilities of the device family. Further deficiencies in the devices were found as a result of this effort. The computer selected for emulation was the 8080. ~~This~~ microprocessor was representative of the higher performance microprocessors on the market, and, as such, was an excellent emulation analysis vehicle.

A major accomplishment of this program was the detailed design of an 8080 emulator ~~utilizing~~ using the device family. The main conclusion that was drawn from this effort is that the device family should be augmented with GUA types with both more capability (gates, pins) and less capability than the TCS-091.

As a supplement (P00001), Questron undertook a functional testing effort of the GPU to verify and validate its design. Questron conceived, developed and proved the concept of low cost functional testing for LSI design verification. This technique utilizes low-cost test stands designed specifically to facilitate computer aided test vector generation and functional testing. During this effort, Questron detected, isolated and diagnosed a total of seven design flaws in the GPU. The GPU is currently being reworked to correct the flaws.

To support the efforts, previously mentioned, Questron developed significant support software capability for the GPU. Microprogram assemblers and a simulator package exist.

To summarize, the major conclusions and accomplishments of this program are as follows:

- A detailed functional specification of the GPU device (Section 1) that will enable potential users of the device to understand its features and also its nuances. Such a specification did not previously exist because of the newness of the device. This functional specification includes design details, control techniques and timing information. The timing specifications take the form of what data paths should be specified – no timing numbers are included since the device has not been characterized by the manufacturer.

- An analysis of the emulation effectiveness of the RCA SOS COS/MOS device family (Section 2). This analysis has resulted in design modifications being recommended for all devices and actual changes being made to the GPU which will significantly enhance its capabilities. In addition, recommendations are made for additional devices for the parts family to upgrade the effectiveness and viability of the family in emulating computers and meeting Air Force requirements.

- Development and proof of the concept of the low-cost functional testing technique for LSI design verification and validation (Section3). This testing technique resulted in the detection, isolation and diagnosis of seven functional design, logic design, electrical design, mask layout and process related flaws. These flaws are currently being ameliorated in a reworked GPU design.

- A detailed design of a breadboard 8080 Emulator was completed (Section 4.) This design is in sufficient detail so that hardware can be procured and a working 8080 Emulator breadboard fabricated.

7 References

1. L. Palkuti and R. Pryor, "Radiation Hard CMOS/SOS Standard Cell Circuits", IEEE Trans. Nuclear Science, Vol. NS-23, December 1976, pp. 1715-1719.

2. G. Skorup, "SOS COS/MOS Universal Array User's Manual", LSI Custom Monolithics Applications Group, Solid State Technology Center, RCA, Somerville, New Jersey, 1 May 1977.

3. J. Brucker, "Transient and Total Dose Radiation Characteristics of a SOS LSI Array", Submitted for publication in the proceedings of the 1977 SOS Workshop.

☆U.S.Government Printing Office: 1978 — 657-002/242