iSBC 957
INTELLEC–iSBC 86/12 INTERFACE
AND
EXECUTION PACKAGE
USER'S GUIDE

Manual Order Number: 9800743A

Printed in U.S.A./B93/0978/5K TL

iSBC 957
INTELLEC–iSBC 86/12 INTERFACE
AND
EXECUTION PACKAGE
USER'S GUIDE

Manual Order Number: 9800743A

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Printed in U.S.A./B93/0978/5K TL

ERRATA

The following information, pertinent to system interfacing, was omitted from Chapter 2 (Installation). Please make the appropriate notations within the chapter to compensate for the omissions.

In section 2-7 (Intellec Series II Model 210) on page 2-2, the following paragraph should be added:

> When interfacing the iSBC 86/12 to the Intellec's Serial 1 port (CRT operation), jumper 51-52 (which connects the RTS output to the CTS input) must be installed on the iSBC 86/12. When interfacing the iSBC 86/12 to the Intellec's Serial 2 port (teletypewriter operation), jumper 51-52 on the iSBC 86/12 must be removed. Note that jumper 51-52 is not installed on units shipped from the factory.

In section 2-8 (Intellec Series II Model 220/230) on page 2-2, the following sentence should be added:

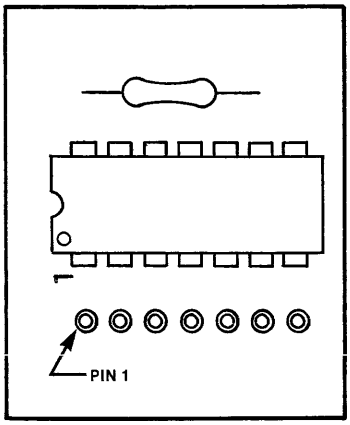
> On the iSBC 86/12, jumper 51-52 (which connects the RTS output to the CRT input) must be installed.

In section 2-9 (Intellec 800 TTY Channel) on page 2-4, the following note should be added to figure 2-3, iSBC 86/12 to Intellec 800 TTY Channel Cabling:

> When installing the TTY UP/DOWN LOAD interface cable (P/N 4002125), the cable connector designated "P1" is inserted into the iSBC 530 TTY adapter, and the cable connector designated "J1" is inserted into the Intellec's TTY channel connector.

Also in section 2-9 and in section 2-10 (Intellec 800 CRT Channel), the following note should be added to the paragraphs describing the installation of additional components:

> When installing the SBC-902 resistor packs, be certain to align pin 1 of the resistor pack (usually denoted by a black dot) with pin 1 of the corresponding socket. When installing the Status Adapter Board assembly (PWA 1002129), be sure that pin 1 of the assembly aligns with pin 1 of socket A11.



TOP VIEW

In section 2-10 (Intellec 800 CRT Channel) on page 2-5/2-6, add the following sentence:

> Jumper 51-52 on the iSBC 86/12 must be removed.

PREFACE

This manual provides general information, interfacing instructions, and programming information for the Intel iSBC 957 Intellec—iSBC 86/12 Interface and Execution Package. Additional information is available in the following documents:

- *Intel 8086 Assembly Language Programming Manual,* Order Number 9800640
- *Intel ISIS-II User's Guide,* Order Number 9800306
- *Intel iSBC 86/12 Single Board Computer Hardware Reference Manual,* Order Number 9800645
- *Intel MCS-86 User's Manual,* Order Number 9800722
- *PL/M-86 Programming Manual*, Order Number 9800466
- *ISIS-II PL/M-86 Compiler Operator's Manual*, Order Number 9800478
- *ISIS-II 8086 Cross Development Utilities Operator's Manual*, Order Number 9800639

CONTENTS

CONTENTS (Cont'd.)

TABLES

ILLUSTRATIONS

CHAPTER 1
GENERAL INFORMATION

1-1. Introduction

The iSBC 957 Intellec–iSBC 86/12 Interface and Execution Package provides the hardware and software required to interface an iSBC 86/12 Single Board Computer with an Intel Intellec Microcomputer Development System.

1-2. Description

The Interface and Execution Package consists of the following:

a. Loader software and monitor ROM's.

b. Four cable assemblies.

c. Teletype adapter.

d. Line drivers and terminators (associated with input/output).

Also supplied are four iSBC 901 Resistor Packs and four type 7437 line driver integrated circuits. These components, which are not used with the implementation of this package, are supplied for the user's own purpose; refer to paragraph 2-10 in the *iSBC 86/12 Single Board Computer Hardware Reference Manual,* Order No. 9800645.

The loader provides the software link between the Intellec system and the iSBC 86/12. The loader is provided on diskettes and operates under ISIS-II control. The cables, teletype adapter, and line drivers and terminators support the different iSBC 86/Intellec configurations. Complete instructions for interfacing the iSBC 86/12 with each of the Intellec models are provided in Chapter 2 and operation information is given in Chapter 3. Programming information is presented in Chapter 4 and the iSBC 86/12 monitor source listing is provided in Appendix A.

1-3. Equipment Supplied

A list of the equipment supplied with the Interface and Execution Package is provided in table 1-1.

Table 1-1. Equipment Supplied

Part Number	Quantity	Description
9500043	1	Single Density Diskette containing ISIS-II iSBC 86/12 Loader
9700039	1	Double Density Diskette containing ISIS-II iSBC 86/12 Loader
9100171	1	I.C., Intel 2716 EPROM (Monitor Program)
9100172	1	I.C., Intel 2716 EPROM (Monitor Program)
9100173	1	I.C., Intel 2716 EPROM (Monitor Program)
9100174	1	I.C., Intel 2716 EPROM (Monitor Program)
4000977	1	iSBC 530 Teletype Adapter
1002129	1	Status Adapter (printed circuit board with 14 pins arranged to plug into integrated circuit socket)
4500644	4**	iSBC 901 Resistor Pack, Pull up/Pull Down
4500645	4**	iSBC 902 Resistor Pack, Pull Up
54-068	4	I.C., 7437, quadruple 2-input positive-NAND buffers
4002127	1*	Cable Assembly, RS232C Up/Down Load (round cable with 25-pin male connector at each end)
4002287	1*	Cable Assembly, Parallel Up/Down Load (flat cable with 50-pin edge connector at one end and an adapter to 25-pin male connector at other end)
4000677	1*	Cable Assembly, OEM RS232C Input/Output (flat cable with 26-pin edge connector at one end and 25-pin RS232C connector at other end)
4002125	1*	Cable Assembly, TTY Up/Down Load (round cable 25-pin male connector at each end)
84-009	8	Screw, panhead, 4-40 x 0.25 inch
9800743	1	iSBC 957 Intellec–iSBC 86/12 Interface and Execution Package User's Guide
9800645	1	iSBC 86/12 Single Board Computer Hardware Reference Manual
9800640	1	Intel 8086 Assembly Language Programming Manual

Note: iSBC 901 Resistor Packs and 7437 IC's are not used in the implementation of the Interface and Execution Package. These parts are supplied to use at your discretion.

* Assembly drawings supplied with cable assemblies.

** Schematic diagrams supplied with resistor packs.

CHAPTER 2
INSTALLATION

2-1. Introduction

This chapter provides information for configuring and interfacing the Interface and Execution Package with the Intellec system. The iSBC 86/12 Single Board Computer must be connected to the spare serial interface channel on the Intellec system. Therefore, the hardware requirement depends on the type (model) of Intellec system and the console device (CRT or teletypewriter) in use.

2-2. General Configuration

The iSBC 86/12 Single Board Computer should be installed in a chassis other than that of the Intellec system. The use of the iSBC 86/12 in the Intellec system chassis is not recommended and is not supported.

2-3. iSBC 86/12 Jumpers/Switches

Except where noted otherwise in this chapter, the general jumper and switch requirements for configuring the ROM/EPROM, Timer Input Frequency (Counter 2), and Serial I/O Port are the default (factory) configurations as defined in the following subparagraphs. Refer to the *iSBC 86/12 Single Board Computer Hardware Reference Manual,* Order No. 9800645, for details.

ROM/EPROM Configuration

The ROM/EPROM default jumpers and switches for the iSBC 86/12 and iSBC 86/12S are as follows:

iSBC 86/12		iSBC 86/12S	
Jumpers	**Switch S1**	**Jumpers**	**Switch S1**
94-96 97-98	8-9 (closed) 7-10 (open)	94-95 98-99	8-9 (closed) 7-10 (open)

Timer (Counter 2) Input Frequency

The 8253 input frequency for Counter 2 (8251 Baud Rate Clock) is 1.23 MHz. This is selected by default jumper 54-55.

Serial I/O Port Configuration

The serial I/O port default jumpers are as follows:

Jumpers In
39-40
42-43
W1 (A-B)
W2 (A-B)
W3 (A-B)

Time Out Option

Inadvertent attempts to access non-existent memory will cause the 8086 CPU to hang up in a wait state. If it is desired to prevent such a hang up, connect jumper 5-6 to enable the failsafe timer. (Jumper 5-6 is *not* connected at the factory.)

Priority Interrupts

The default (factory installed) jumpers (posts 74 through 81) connecting interrupt inputs to the 8259A Programmable Interrupt Controller (PIC) should be removed unless the user program is expecting them.

2-4. iSBC 86/12 Monitor Program

The Monitor Program resides in the four 2716 EPROM's. Install these four chips in the iSBC 86/12 as follows:

Part No.	Socket
9100171	A28
9100172	A46
9100173	A29
9100174	A47

2-5. Intellec System Jumpers

Ensure that any I/O port (channel) of the Intellec system to be connected to the iSBC 86/12 has all default (factory) jumpers connected. Refer to the applicable Intellec system manual. During its initialization sequence, the loader reprograms the Intellec USART to the iSBC 86/12 for 9600 baud and assumes the jumpers are configured as shipped from the factory. (Exception: The Intellec 800 TTY channel is *not* programmed.)

2-6. System Interfacing

The following paragraphs describe the interfacing required between the iSBC 86/12 and the various Intellec systems.

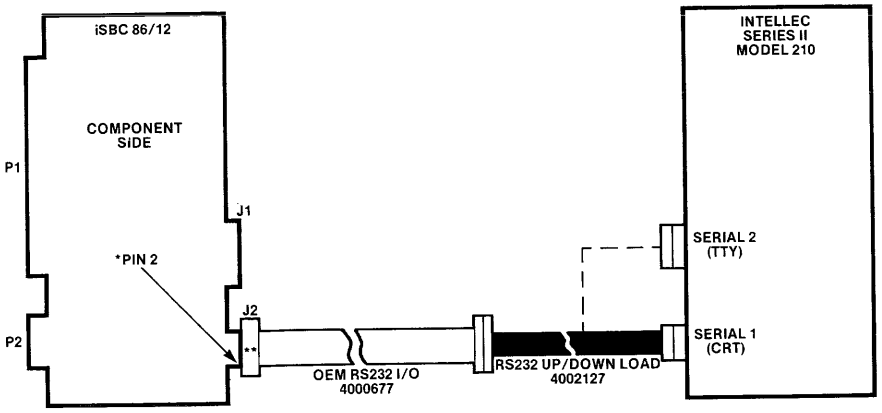
2-7. Intellec Series II Model 210

The Intellec Series II Model 210 must be upgraded with disk drives in order to use it with the Interface and Execution Package.

The iSBC 86/12 interfaces to the Serial 1 or Serial 2 port (channel) of the Intellec Series II Model 210. If the Intellec console is a CRT device, the Serial 1 port is used; if the console is a teletypewriter, the Serial 2 port is used. Connect the cables as shown in figure 2-1. Secure the RS232C connector to the Intellec chassis using two 4-40 panhead screws.

2-8. Intellec Series II Model 220/230

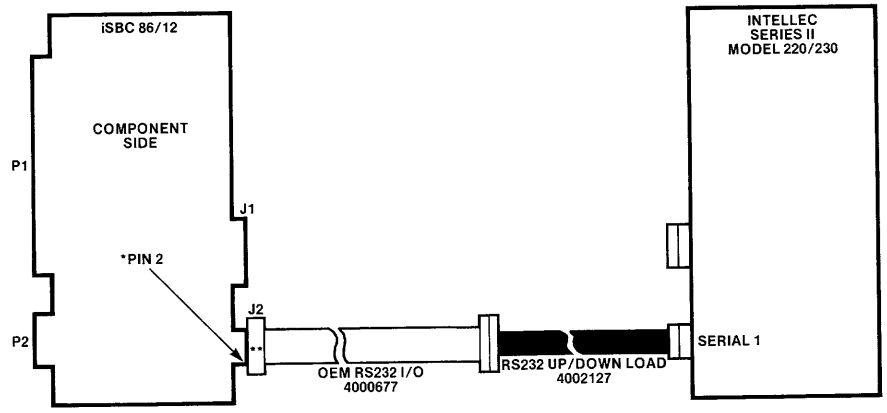
The iSBC 86/12 interfaces to the Serial 1 Port of the Intellec Series II Model 220 or Model 230. Connect the cables as shown in figure 2-2. Secure the RS232C connector to the Intellec chassis using two 4-40 panhead screws.



*PIN 2 OF iSBC 86/12 IS ON SOLDER SIDE OF BOARD.

**INSTALL CABLE SO THAT PIN 1 ON CABLE CONNECTOR MATES WITH PIN 2 ON iSBC 86/12 EDGE CONNECTOR.

Figure 2-1. iSBC 86/12 to Intellec Series II Model 210 Cabling 743-1



*PIN 2 OF iSBC 86/12 IS ON SOLDER SIDE OF BOARD.

**INSTALL CABLE SO THAT PIN 1 ON CABLE CONNECTOR MATES WITH PIN 2 ON iSBC 86/12 EDGE CONNECTOR.

Figure 2-2. iSBC 86/12 to Intellec Series II Model 220/230 Cabling 743-2

2-9. Intellec 800 TTY Channel

The iSBC 86/12 interfaces to the TTY channel of the Intellec 800 when the Intellec console is a CRT device. In this configuration, there are two operating modes available. One mode uses the iSBC 86/12 serial I/O port only; the other mode uses the iSBC 86/12 serial I/O port and a parallel interface.

The parallel interface provides a slightly higher transmission rate for loading files than is possible with the Intellec 800 CRT channel and, therefore, the parallel interface is used only with the load and transfer commands. This requires the use of the 8255A Programmable Peripheral Interface on the iSBC 86/12.

To provide the proper electrical connection (protective ground) for the serial I/O port, install a jumper between posts 63 and 64 (near edge connector J2) and the iSBC 86/12.

The iSBC 86/12 requires the installation of additional components and the modification of default (factory configured) jumpers if the parallel interface is used. Install the components and change the default jumpers as described below.

Install Component	Location
iSBC 902 Resistor Pack	A10, A12, A13
Status Adapter Board	A11

Remove Jumper	Add Jumper
21-25	25-31
13-14	18-31
32-33	14-30
26-27	20-33
19-20	13-27
30-31	

After the iSBC 86/12 is configured, connect the cables as shown in figure 2-3. Secure the connectors to the iSBC 530 TTY Adapter chassis using four 4-40 panhead screws. (Two screws per connector.) Secure the connectors to the Intellec chassis using four 4-40 panhead screws. (Two screws per connector.)

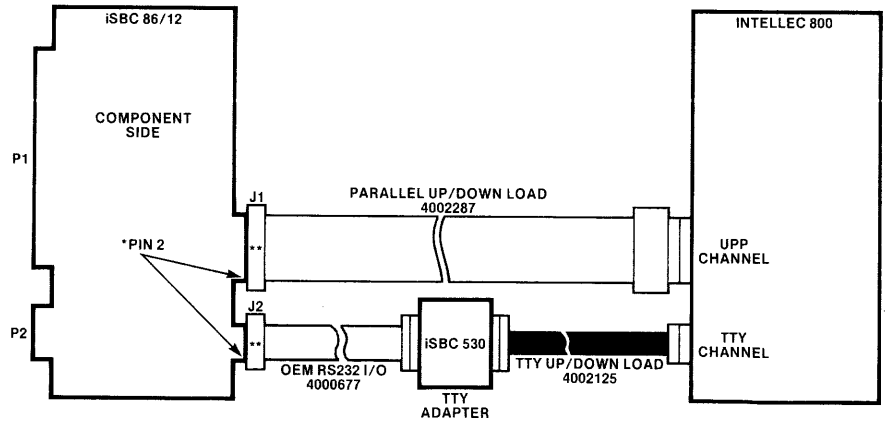


Figure 2-3. iSBC 86/12 to Intellec 800 TTY Channel Cabling 743-3

2-10. Intellec 800 CRT Channel

The iSBC 86/12 is interfaced to the CRT channel of the Intellec 800 when the Intellec console is a teletypewriter. In this configuration there are two operating modes available. One mode uses the iSBC 86/12 serial I/O port only; the other mode uses the iSBC 86/12 serial I/O port and a parallel interface.

The parallel interface provides a much faster transmission rate for loading files than is possible with the Intellec 800 TTY channel and, therefore, the parallel interface is used only with the load and transfer commands. This interface requires the use of the 8255A Programmable Peripheral Interface on the iSBC 86/12.

The iSBC 86/12 requires the installation of additional components and the modification of default (factory configured) jumpers if the parallel interface is used. Install the components and change the default jumpers as described below.

Install Component	Location
iSBC 902 Resistor Pack	A10, A12, A13
Status Adapter Board	A11

Remove Jumper	Add Jumper
21-25	25-31
13-14	18-31
32-33	14-30
26-27	20-33
19-20	13-27
30-31	

After the iSBC 86/12 is configured, connect the cables as shown in figure 2-4. Secure the connectors to the Intellec chassis using four 4-40 panhead screws. (Two screws per connector.)

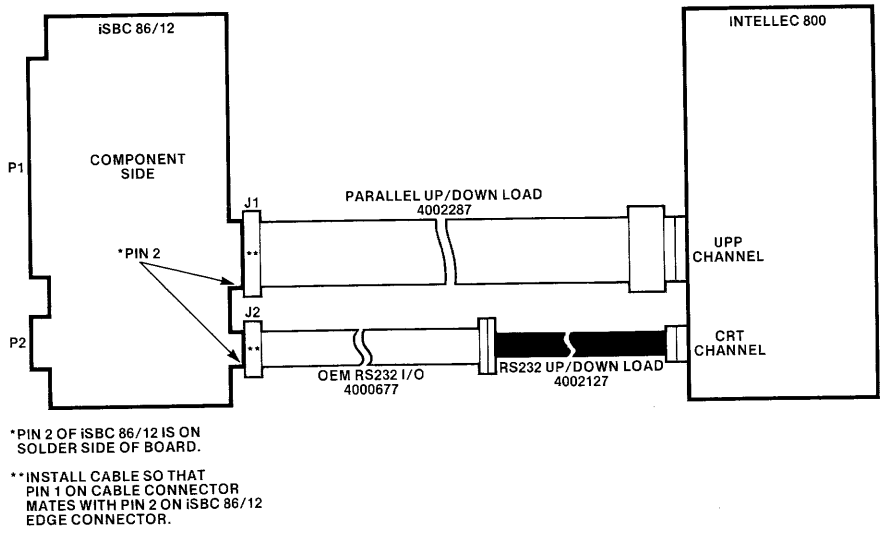


Figure 2-4. iSBC 86/12 to Intellec 800 CRT Channel Cabling 743-4

CHAPTER 3
OPERATION

3-1. Introduction

This chapter provides operating instructions for the Interface and Execution Package The loader provides commands for loading programs into the iSBC 86/12 from an Intellec system and for transferring programs from the iSBC 86/12 to an Intellec system. The loader also supports the debug commands provided by the iSBC 86/12 on-board monitor.

3-2. Start-Up Procedure

The iSBC 86/12 loader program requires an Intellec system with 32K bytes of memory. The loader program is loaded from ISIS-II by invoking the program SBC861 when ISIS prompts for a command as follows:

```
-:Fn:SBC861
ISIS-II ISBC 86/12 LOADER, Vx.x
```

where :Fn: specifies the drive number and x.x denotes the current version of the loader program.

After the sign-on message appears, power-on or reset the iSBC 86/12. Then type the character "U" on the keyboard twice in order for the on-board monitor to determine the baud rate of the interface. When the baud rate has been successfully determined, the monitor sign-on message

```
ISBC 86/12 MONITOR, Vx.x
```

is output to the console. When the monitor is ready, it will prompt with a period "." at the beginning of a new line.

NOTE

If random characters are displayed, the incorrect baud rate was determined by the on-board monitor. In this case, reset the iSBC 86/12 and again type the character "U" twice.

3-3. Command Structure

The loader prompts with a period "." when it is ready for a command. You can then enter a command line, which consists of a one- or two-character command followed by zero, one, or more arguments. The command may be separated from the first argument by an optional single space; a single comma is required as a delimiter between arguments. The command line is terminated by a carriage return or a comma depending on the command, and no action takes place until the command terminator is sensed. You can cancel a command before entering the command terminator by pressing any illegal key (e.g., rubout or Control-X).

Only uppercase letters and digits may be used in the commands and arguments. Numeric arguments can be expressed as a number, the contents of a register, or the sum or difference of numbers and register contents. Thus, addresses and data can be expressed as follows:

```
<addr>::=     [<expr>:]<expr>
<expr>::=     <number>|<register>|<expr> {+|-} <number>|
              <expr> {+|-} <register>
<register>::= AX|BX|CX|DX|SP|BP|SI|DI|CS|DS|SS|ES|IP|FL
<number>::=   <digit>|<digit><number>
<digit>::=    0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F
```

Numeric fields within arguments are entered as hexadecimal numbers. The valid range of numerical values is from 0000-FFFF. Larger numbers may be entered, but only the last four digits (or two in the case of byte values) are significant. Leading zeros may be omitted.

Whenever word values are displayed, the contents of the high address location is displayed followed by the contents of the low address location. Similarly, when entering word values, the high byte is followed by the low byte. If necessary leading zeros will be appended to the value by the monitor.

An address argument consists of a segment value and an offset value separated by a colon (:). If a segment value is not specified, the default segment value is the CS register value except where noted otherwise in the command description.

NOTE

Since the commands are not line oriented, the loader *cannot* execute under the ISIS-II SUBMIT facility.

3-4. Errors

Each character input to the monitor is checked for validity. An erroneous entry causes the monitor to type a "#" and prompt for a new command by outputting "." on a new line. The monitor detects an attempt to modify EPROM or non-existent RAM by writing the data, reading it back, and then comparing the values. If the iSBC 86/12 board is not jumpered to provide a timeout when non-existent RAM is accessed, then the system will hang. (Refer to Chapter 2.)

If an error occurs when attempting to access an ISIS-II file, the error number is displayed and the command is aborted. After a non-fatal error, the monitor prompts for a new command. A fatal error results in a return to ISIS.

3-5. Control Characters

Control characters are used to start and stop the output or execution of the D, X, C, F, and W commands. Control-S is used to temporarily stop the operation of a command; Control-Q is used to resume the operation of the command. Control-C is used to abort the operation. After a Control-S, the only acceptable characters are Control-Q and Control-C.

3-6. 8086 CPU Registers

The 8086 CPU, which is the heart of the iSBC 86/12, includes the 14 registers listed in table 3-1. These registers are referenced in the command syntax in their abbreviated form.

Table 3-1. 8086 CPU Registers

Register Name	Abbreviation
Accumulator	AX
Base	BX
Count	CX
Data	DX
Stack Pointer	SP
Base Pointer	BP
Source Index	SI
Destination Index	DI
Code Segment	CS
Data Segment	DS
Stack Segment	SS
Extra Segment	ES
Instruction Pointer	IP
Flag	FL

3-7. Command Descriptions

This section describes the 14 loader and monitor commands listed in table 3-2. The syntax conventions used in the command structure are as follows:

- [A] indicates that "A" is optional
- [A]* indicates one or more optional iterations of "A"
- <B> indicates that "B" is a variable
- {A|B} indicates "A" or "B"
- <cr> indicates a carriage return is entered

The following paragraphs describe the 14 commands and provide error conditions and examples where appropriate. In the examples, the loader/monitor outputs are underscored.

Table 3-2. Monitor Command List

Command		Function and Syntax
L	Load Hex Object File	Loads hexadecimal object file from Intellec into iSBC 86/12 memory. L {S|P} ,<*filename*>[,<*bias addr*>]<*cr*>
T	Transfer Hex Object File	Transfers block of iSBC 86/12 memory to Intellec as a hex object file. T[X] {S|P},<*start addr*>,<*end addr*>,<*filename*> [,<*exec addr*>]<*cr*>
E	Exit	Exits the loader program and returns to ISIS. E<*cr*>
N	Single Step	Executes one user program instruction. N[<*addr*>],[[<*addr*>],]*<*cr*>
G	Go	Transfers control of the 8086 CPU to the user program. G[<*start addr*>][,<*break 1 addr*> [,<*break 2 addr*>]]<*cr*>

Table 3-2. Monitor Command List (Cont'd.)

Command		Function and Syntax
S	Substitute Memory	Displays/modifies memory locations S[W]<*addr*>,[[*new contents*],]*<*cr*>
X	Examine/Modify Register	Displays/modifies 8086 CPU registers. X[<*reg*>][[<*new contents*>],]*<*cr*>
D	Display Memory	Displays contents of a memory block. D[W]<*start addr*>[,<*end addr*>]<*cr*>
M	Move	Moves contents of a memory block. M<*start addr*>,<*end addr*>,<*destination addr*><*cr*>
C	Compare	Compares two memory blocks. C<*start addr*>,<*end addr*>,<*destination addr*><*cr*>
F	Find	Searches a memory block for a constant. F[W]<*start addr*>,<*end addr*>,<*data*><*cr*>
H	Hex Arithmetic	Performs hexadecimal addition and subtraction. H<*data 1*>,<*data 2*><*cr*>
I	Port Input	Inputs and displays data from input port. I[W]<*port addr*>,[,]*<*cr*>
O	Port Output	Outputs data to output port. O[W]<*port addr*>,<*data*>[,<*data*>]*<*cr*>

3-8. Load Hexadecimal File (L)

Function

This command (L) loads a hexadecimal object file into the iSBC 86/12 system from the Intellec system.

Syntax

L $\left\{ \text{S|P} \right\}$,<*filename*>[,<*bias addr*>]<*cr*>

where S=serial mode and P=parallel mode.

Operation

The first argument $\left\{ \text{S|P} \right\}$ specifies which interface is used: serial (S) or parallel (P). The parallel mode is not supported on an Intellec Series II. The user then enters the *filename* specifying a file in the 8086 or 8080 hexadecimal object file format. The data read from the file is output to the iSBC 86/12 where it is loaded into the locations specified by the extended address records and load address fields. If the file is in 8080 format, the data is loaded into memory locations relative to 0.

If an optional *bias addr* is specified, its segment value is added to each extended address record in the file and its offset value is added to each offset address in the file, thus forming a new load address. For an 8080 file, the segment value specified is added to the default segment value 0. If no segment value is specified in the *bias addr,* then 0 is used as the segment value for the bias.

If the file is in 8086 format and includes an 8086 execution start address record, then the CS and IP registers are set to the values specified in the record. However, if the file is in 8080 format and the end-of-file record includes an 8080 execution start address, then the IP register is set to this value and the CS register remains unchanged. Thus, a subsequent Go (G) or Single Step (N) command uses the address specified in the file unless modified by the user.

Error Conditions

The following error conditions are associated with the Load (L) command:

(1) Attempt to store into EPROM or non-existent RAM.

(2) ISIS-II type (1-99) errors.

(3) "P" specified in first argument on Intellec Series II.

(4) Timeout.

(5) Checksum error.

Example

Load file :F1:ABC.HEX into iSBC 86/12 memory using the serial interface:

```
-L S,:F1:ABC.HEX<cr>
```

3-9. Transfer Hexadecimal File (T)

Function

This command (T) transfers the contents of a block of memory in the iSBC 86/12 system to a hexadecimal object file in the Intellec system.

Syntax

T[X] {S|P} ,<*start addr*>,<*end addr*>,<*filename*>[,<*exec addr*>]<*cr*>

where T=8086 format and TX=8080 format.

Operation

The first argument {S|P} specifies which interface is used: serial (S) or parallel (P). The parallel mode is not supported on the Intellec Series II. The source locations are specified by the *start addr* and *end addr.* The destination is specified as the *filename* of an ISIS-II file. The on-board monitor outputs the data in 8086 or 8080 hexadecimal object file format. The user may optionally enter the *exec addr* as the address to be put in an 8086 execution start address record or the 8080 end-of-file record.

If the 8086 format is specified, then the segment value for the first extended address record and the offset value for the load address field of the first data record are taken from the corresponding fields within the *start addr* expression. If no segment value is specified in the *end addr,* then it defaults to the value specified or implied in the *start addr.* The CS and IP register values for the execution start address record are taken from their respective fields within the *exec addr* expression.

If the 8080 format is specified, then the load address for the first data record is taken from the offset value in the *start addr* expression, although the data is output from the actual address entered. The execution address put in the end-of-file record is taken from the *exec addr* if specified. No segment value is allowed in the *end addr* and *exec addr.*

Error Conditions

The following error conditions are associated with the Transfer (T) command:

(1) *end addr* is less than *start addr.*

(2) ISIS-II type (1-99) errors.

(3) "P" specified in first argument on Intellec Series II.

(4) Timeout.

(5) Checksum error.

Example

Transfer iSBC 86/12 memory between 00400 and 0FE50, relative to current CS register, into file :F2:MYPROG in 8086 format using the serial interface:

```
-T S,400,FE50,:F2:MYPROG<cr>
```

3-10. Exit (E)

Function

This command (E) allows you to exit the loader and return to ISIS.

Syntax

E<*cr*>

3-11. Single Step (N)

Function

This command (N) executes one user program instruction.

Syntax

N[<*addr*>],[[<*addr*>],]*<*cr*>

Operation

After "N" is entered, the monitor displays the current IP register contents followed by a "–" and the instruction byte pointed to by the IP register. If you wish to modify the IP register or both the CS and IP registers, enter an *addr* value followed by a comma. If the *addr* value includes a segment value, then both the CS and IP registers are modified. Otherwise, only the IP register is modified.

A comma, when typed after the next instruction byte is displayed or after entering *addr,* causes all user registers to be restored and a single instruction within the user program to be executed. The monitor is then reentered, all user registers are saved, and the new IP (followed by "–" and the next instruction byte) is displayed. The procedure for modifying the IP (and CS) register and/or single-stepping the next instruction byte can be repeated any number of times. A carriage return after the next instruction byte is displayed terminates the command.

The following restrictions apply to the single-step command:

(1) If an interrupt occurs prior to the completion of the single-stepped instruction, or if the single-stepped instruction generates an interrupt (e.g., an INT instruction), then upon reentering the monitor, the CS and IP register values will point to the interrupt service routine. An exception occurs with the INT 3 instruction. In this case, the CS and IP values will point to the instruction after the INT 3.

(2) An instruction that is part of a sequence of instructions that switches stacks (such that the new stack is in a new segment) cannot be single stepped.

Examples

(1) .N 3C07– 40 3C00	Change IP to 3C00 and step.
(2) 3C01– 5B,	Step.
(3) 3C02– 5A,	Step.
(4) 3C03– 5B<*cr*>	Terminate command.

3-12. Go (G)

Function

This command (G) transfers control of the CPU from the monitor to the user program.

Syntax

G[<*start addr*>][,<*break 1 addr*>[,<*break 2 addr*>]]<*cr*>

Operation

After "G" is entered, the monitor displays the current IP register contents followed by a "–" and the instruction byte pointed to by the IP register. If you wish to modify the IP register or both the CS and IP registers, enter the *start addr.* When the *start addr* includes a segment value, both the CS and IP registers are modified. Otherwise, only the IP register is modified.

One or two optional breakpoint addresses may be specified by entering a comma followed by *break 1 addr* and *break 2 addr* after the instruction byte is displayed or after entering the *start addr.* If a segment value was specified in the *start addr,* then it becomes the default segment value for each *break addr.*

A carriage return after the optional arguments are entered or after the next instruction byte is displayed causes the monitor to restore all user registers and pass control to the user program.

Breakpoints are implemented by replacing each breakpointed instruction with an INT 3 instruction, and therefore only instructions in RAM may be breakpointed. Upon execution of the INT 3 instruction, the monitor is reentered, all registers are saved, the breakpointed instruction(s) are restored, "BRn @XXXX:YYYY ZZ" is displayed (where n = 1 or 2, XXXX = CS, YYYY = IP, and ZZ = next instruction byte), and you are prompted for a new command. If the monitor is reentered through an internally or externally generated interrupt before a breakpoint occurs, the breakpointed instruction(s) is restored by the monitor.

Error Conditions

An error is caused by an attempt to breakpoint EPROM or non-existent RAM.

Examples

(1) .G 1000- F5<*cr*>	Begin at current IP.
(2) .G 1000- F5 3000<*cr*>	Begin at 3000 relative to current CS.
(3) .G 1000- F5 0015:3000<*cr*>	Begin at 3000 relative to CS value of 0015.
(4) .G 1000- F5 3000, 3C01<*cr*> BR1 @0015:3C01 F5	Begin at 3000 and set breakpoint 1 at 3C01. Monitor is entered at breakpoint.
(5) .G 1000- F5 ,3C01<*cr*>	Begin at current IP and set breakpoint at 3C01.

3-13. Substitute Memory (S)

Function

This command (S) allows you to display and optionally modify memory locations on a byte or word basis.

Syntax

S[W]<*addr*>,[[<*new contents*>],]*<*cr*>

where S=byte mode and SW=word mode.

Operation

The memory contents are displayed and modified as bytes if "S" is specified; the memory contents are displayed and modified as words if "SW" is specified. When the command is entered, the contents of the memory location specified by *addr* is displayed followed by the prompt "–".

If you wish to modify the contents of the location displayed, enter the *new contents* (data to be stored) followed by a comma or a carriage return. The location is then updated with the new data entered.

If a comma is typed after the monitor's prompt or after entering the *new contents,* the offset address and contents of the next location are displayed on a new line followed by another prompt "–". The procedure for modifying the current location and/or displaying the next location can then be repeated any number of times. A carriage return after any prompt "–" or after the entry of *new contents* terminates the command.

Error Conditions

An error is caused if the *addr* is located in EPROM or in non-existent RAM.

Examples

(1) Examine locations 3FF0 to 3FF3, relative to the DS register, and modify location 3FF2:

```
.S DS:3FF0, F5- ,
3FF1 40- ,
3FF2 59- 5D,
3FF3 C3- <cr>
```

(2) Examine and modify top element of stack:

```
.SW SS:SP, 3C1F- 3C1E<cr>
```

3-14. Examine/Modify Register (X)

Function

This command (X) allows you to examine and optionally modify the CPU registers.

Syntax

X[<*reg*>][[<*new contents*>],]*<*cr*>

Operation

This command (X) has two forms depending on whether or not a register is specified in the *reg* variable. In the first form, when a register is specified (e.g., AX, BX, SI, CS, etc.), the contents of the specified register are displayed followed by a prompt "–". If you wish to modify the contents of the register, enter the *new contents* followed by a comma or carriage return. The register is then updated with the new data entered. If a comma is typed after the monitor's prompt or after entering the *new contents* value, the name and contents of the next register in order is displayed on a new line followed by another prompt "–". (The registers are displayed in the sequence listed in table 3-1.) The procedure for modifying the current register contents and/or displaying the next register can be repeated until the last register has been displayed. A carriage return after any prompt "–", or after entry of a *new contents* value, terminates the command.

In the second form, no register is specified and the command is simply X followed by a carriage return. This form of the command displays all the CPU register names and contents in the order listed in table 3-1.

Examples

(1) Modify the contents of the SI register and then examine the DI register:

```
.X SI=034C- FFF,
DI=F638- <cr>
```

(2) Examine all the CPU registers:

```
.X<cr>
AX=FFFF BX=FFFF CX=FFFF DX=FFFF SP=FFFF BP=FFFF SI=FFFF
DI=FFFF CS=FFFF DS=FFFF SS=FFFF ES=FFFF IP=FFFF FL=FFFF
```

3-15. Display Memory (D)

Function

This command (D) displays the contents of a specific block of memory; the contents can be displayed on a byte or word basis.

Syntax

D[W]<*start addr*>[,<*end addr*>]<*cr*>

where D=byte mode and DW=word mode.

Operation

This command provides a line-by-line formatted hexadecimal display of the memory block bounded by the *start addr* and *end addr,* inclusive. The *end addr* is an offset address relative to the segment value specified or implied in the *start addr* expression; consequently, no segment value is allowed. This means that the block size is limited to 64K bytes. If *end addr* is omitted, only one location is displayed. The display can be as bytes or words, depending upon whether an optional "W" is entered. In the word mode, the high-order byte is followed by the low-order byte. Each line of the display begins with the offset address of the first memory location displayed on that line, followed by the contents of each location (or pair of locations in the word mode).

Error Condition

An error is caused if the *end addr* is less than the offset value of the *start addr.*

Examples

(1) Display locations 0009 through 002A relative to DS register:

```
.D DS:9,2A<cr>
0009 00 00 00 00 00 00 00
0010 34 12 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00
```

(2) Display location 0019 relative to CS register:

```
.D 19<cr>
0019 FE
```

(3) Display locations 0009 through 002A, relative to DS register, in word mode:

```
.DW DS:9,2A<cr>
0009 0000 0000 0000 3400
0011 0012 0000 0000 0000 0000 0000 0000 0000
0021 0000 0000 0000 0000 0000
```

3-16. Move (M)

Function

This command (M) moves the contents of a memory block.

Syntax

M<*start addr*>,<*end addr*>,<*destination addr*><*cr*>

Operation

The move command moves the contents of the memory block bounded by the *start addr* and *end addr,* inclusive, to the memory locations starting at the *destination addr.* The *end addr* is an offset address relative to the segment value specified or implied in the *start addr* expression, and consequently no segment value is allowed. This means that the block size is limited to 64K bytes. The move is done a byte at a time beginning with the contents of the *start addr.* When the move is completed, the monitor prompts for a new command.

The move command may be used to fill memory with a constant by first modifying the contents at location *start addr* with the constant, and then specifying the *destination addr* as *start addr* + 1 in a move command. Note that, in this case, the specified *end addr* must be one less than the actual end address desired.

Error Conditions

The following error conditions apply to this command:

(1) *end addr* is less than offset value of *start addr.*

(2) Attempt to move block to EPROM or non-existent RAM.

Example

Move the contents of memory between locations 0200 and 0250H, relative to DS register, to the destination address with segment value of ES + 10 and offset value of 0150:

```
_M DS:200,250,ES+10:150<cr>
```

3-17. Compare (C)

Function

This command (C) compares the contents of one block of memory with that of another block.

Syntax

C<*start addr*>,<*end addr*>,<*destination addr*><*cr*>

Operation

This command compares the contents of the memory block bounded by the *start addr* and *end addr,* inclusive, with the contents of the memory block starting at the *destination addr.* The *end addr* is an offset address relative to the segment value specified or implied in the *start addr* expression, and consequently no segment value is allowed. This means that the block size is limited to 64K bytes. Each time the contents of a location in the first block is not equal to the contents of the corresponding

location in the destination block, the mismatch is displayed. A single line is formatted with the offset address and contents of the location within the first block, followed by the offset address and contents of the location within the destination block.

Error Condition

An error occurs if the *end addr* is less than the offset value of the *start addr.*

Example

Compare the contents of memory between locations 0000 and 0030, relative to CS register, with the contents of memory starting at 0050H, relative to ES register:

```
.C 0,30,ES:50<cr>
0021 00 0071 1F
002D C8 007D D3
```

3-18. Find (F)

Function

This command (F) searches a block of memory for a match on a byte or word value.

Syntax

F[W]<*start addr*>,<*end addr*>,<*data*><*cr*>

where F=find byte and FW=find word.

Operation

The search is performed in a sequential manner over the block bounded by the *start addr* and *end addr,* inclusive. The *end addr* is an offset address relative to the segment value specified or implied in the *start addr* expression, and consequently no segment value is allowed. This means that the block size is limited to 64K bytes. Each time a match is found, its offset address is displayed. When the entire block has been searched, the monitor prompts for a new command.

In the word mode after each comparison is made, the monitor's address pointer for the next comparison is incremented by one as shown in the example below.

Error Condition

An error occurs if the *end addr* is less than the offset value of the *start addr.*

Example

Search locations 0100-0101, 0101-0102, 0102-0103, 0103-0104, relative to DS register, for FFFF.

```
.FW DS:100,103,FFFF<cr>
0100
0103
```

The contents of locations 0100-0101 and 0103-0104 equal FFFF.

3-19. Hexadecimal Arithmetic (H)

Function

This command (H) performs hexadecimal addition and subtraction of two arguments.

Syntax

H<*data 1*>,<*data 2*><*cr*>

Operation

Two's complement arithmetic modulo 2^{16} is performed. After the carriage return is entered, the monitor prints the sum (value of *data 1* plus the value of *data 2*), a space, and the difference (value of *data 1* minus the value of *data 2*).

Example

```
.H 0C52,0401<cr>
1053 0851
```

3-20. Port Input (I)

Function

This command (I) inputs and displays a byte or a word from the specified port.

Syntax

I[W]<*port addr*>,[,]*<*cr*>

where I=input byte and IW=input word.

Operation

This command inputs a byte or word from the port specified by the *port addr* and displays the value. No segment value is allowed in the *port addr* expression since the I/O space is addressed as if it were a single segment. A comma must be entered after the *port addr* to activate the command. After the input value is displayed, a comma may be entered again to perform another input operation from the same port. This procedure may be repeated any number of times. A carriage return terminates the command.

Example

Input a byte from port F7 three times:

```
.I F7,
2E,
2E,
2F<cr>
```

3-21. Port Output (O)

Function

This command (O) outputs a byte or a word to the specified port.

Syntax

O[W]<*port addr*>,<*data*>[,<data>]*<*cr*>

where O=output byte and OW=output word.

Operation

This command outputs the byte or word specified by the *data* to the port specified by *port addr.* No segment value is allowed in the *port addr* expression since the I/O space is addressed as if it were a single segment. If more than one output operation to the same port is to be executed, a comma may be typed after *data.* The monitor will output the value to the specified port and display "–" on a new line, prompting for the next value to be output. The procedure may be repeated by entering another *data* followed by a comma. A carriage return entered after a prompt terminates the command. A carriage return after entering *data* outputs the value and then terminates the command.

Example

Output 3C5, 31, 32, 34 to port FA.

```
.OW FA, 3C5,
-31,
-32,
-34<cr>
```

CHAPTER 4
PROGRAMMING INFORMATION

4-1. Introduction

The memory organization, initialized contents of the 8086 CPU registers, USART baud rate requirement, interrupt servicing, and instruction breakpoints are discussed in the following paragraphs.

4-2. Memory Organization

The RAM storage area from 0H to 17FH is reserved for the monitor stack, the monitor data area, and interrupt vectors. The monitor's constants and code reside in EPROM locations FE800H to FFFFFH. (EPROM locations FE000H to FE7FFH are not used.) The user may utilize RAM locations from 1C0H to FDFFFH as desired. During start-up, the monitor sets the user's stack pointer to 001C0H, pointing to the first available location in the user's RAM area above the initial user's stack, for which 64 bytes are reserved (locations 180H to 1BFH). This may not be enough for a user application, in which case the user's initialization code should change the stack pointer's value.

The monitor reserves locations 0H to 9FH for forty 4-byte interrupt vectors. Vectors 0 and 4 have dedicated hardware functions. Vectors 1, 2, and 3 are reserved by the monitor for single-step, non-maskable interrupt, and breakpoint. Vectors 5-31 are reserved for future use by Intel. Vectors 32-39 are used for the 8259A interrupt controller. Vectors 112 to 255 (locations 1C0H to 3FFH) above the monitor's data area is available to the user.

Whenever the monitor is reentered (single step, breakpoint, interrupt), it temporarily uses 13 words of the user's stack to save registers. This value must be taken into consideration when allocating memory for the stack. The registers are removed from the stack before the prompt is issued.

4-3. 8086 CPU Register Initialization

When power is initially applied to the iSBC 86/12, or when reset, the monitor initializes the user's 8086 register values as follows:

CS = 0000
SS = 0000
DS = 0000
ES = 0000
IP = 0000
FL = 0000
SP = 01C0H

4-4. USART Initialization

When power is initially applied to the iSBC 86/12, or when reset, the monitor automatically programs the 8251A USART for interface with the console device. The USART is configured to the following state:

a. Mode:
   1 stop bit at 150-9600 baud
   2 stop bits at 110 baud
   Parity disabled
   8 bit character length
   Baud rate factor of 16X

b. Command:
   Request-To-Send
   Error reset
   Receiver enabled
   Data-Terminal-Ready at 150-9600 baud
   Transmitter enabled

NOTE

Counter 2 of the 8253 Programmable Interval Timer is used to establish the baud rate. Care must be exercised by the user in modifying the USART mode and command since the monitor's device drivers depend on the configuration defined above for proper operation. Also, the mode of Counter 2 of the 8253 should not be modified.

4-5. Interrupt Servicing

When power is initially applied to the iSBC 86/12, or when reset, the monitor sets (1) the single-step interrupt vector, (2) the non-maskable interrupt vector, and (3) the one- byte trap instruction interrupt vector to monitor entry points.

The monitor also sets the 8259A Programmable Interrupt Controller to the fully nested mode with level 0 at the highest priority. All interrupts are unmasked. The 8259A is initialized to provide vectored interrupts through a table located in RAM at 80H. All entries are initialized to reenter the monitor though a special interrupt entry point. When an interrupt occurs, control is returned to a monitor routine that saves the user's registers. It then displays the interrupt level, the CS and IP register values (that were saved by the interrupt sequence of the hardware), and the instruction byte pointed to by the IP register. The monitor then acknowledges the interrupt and prompts the user for a command as follows:

```
.G 3000<cr>
I=3 @1000:230F F5
.
```

The Go (G) command may be used to resume operation.

The user may also elect to field his own interrupts. In that case, for each interrupt two words of the table should be modified to contain the address of the user's interrupt handler. The IP value is stored at

(80H + 4*INTERRUPT#)

and the CS value is stored at

(82H + 4*INTERRUPT#)

Interrupts are disabled during user/monitor command interaction so that pending interrupts will not interfere with program checkout. The user program's interrupt state is restored on exiting the monitor via the Go (G) or Single Step (N) command.

4-6. Instruction Breakpoints

If the user codes an INT 3 instruction into his program, the monitor is reentered when that instruction is executed. The monitor saves all registers and then prints an "@", the contents of the CS and IP registers, and the instruction byte pointed to by the IP (which is the instruction after the INT 3). The monitor then prompts the user for a command. Program execution may be resumed with the Go (G) or Single Step (N) command.

Example

The user program contains an INT 3 instruction at location 3F02, relative to CS segment value 1200. Upon execution of the INT 3 the monitor displays the following:

```
@1200:3F03 F5
-
```

4-7. System I/O Routines

The diskette on which the loader resides also includes two libraries containing I/O routines for the console. The routines in both libraries have the same names and functions, but two libraries are necessary to support the two types of subroutine linkages provided by the 8086 architecture. The routines in SBCIOS.LIB are written to be called with intrasegment subroutine calls. A PL/M-86 module compiled with the "small" control generates this type of call. The routines in SBCIOL.LIB are written to be called with intersegment subroutine calls. A PL/M-86 module compiled with either the "medium" or "large" control generates this type of call.

The routines in both libraries were written in PL/M-86. The modules in SBCIOS.LIB were compiled with the "small" control. The modules in SBCIOL.LIB were compiled with the large control. The names assigned to the segments, classes, and groups are the standard names generated by the PL/M-86 compiler. (See *ISIS-II 8086 Cross Development Utilities Operator's Manual,* Order No. 9800639).

The console input and output routines provided in the library should be used for console I/O. The loader has some special requirements which are handled by the routines described in the following paragraphs.

4-8. Console Input (CI) Routine

This routine returns an 8-bit character received from the console device to the caller in the AL register. The AX, CX, and DX registers and the CPU condition codes are affected by this operation.

If a Control-S (13H) or Control-Q (11H) character is read, then the character is "thrown away" and another is read. This is necessary in order to operate properly with the feature provided in the CO routine.

Example 1

```
PL/M-86 CI Call Example

CI: PROCEDURE BYTE EXTERNAL;
  END CI;

DECLARE CHAR BYTE;
     .
     .
     .
CHAR = CI AND 7FH;     /* INPUT CHARACTER AND STRIP PARITY BIT */
     .
     .
     .
```

Example 2

```
ASM86 CI Intrasegment Call Example

          ASSUME  DS:DATA,SS:STACK,CS:CODE

DATA      SEGMENT PUBLIC 'DATA'
CHAR      DB      ?
DATA      ENDS

STACK     SEGMENT STACK 'STACK'
          DW      15 DUP ?
BASESTACK LABEL   WORD
STACK     ENDS

CODE      SEGMENT PUBLIC 'CODE'
          EXTRN   CI:NEAR
INIT:
          MOV     AX,STACK              ; INITIALIZE
          MOV     SS,AX                 ;   SS
          MOV     SP,OFFSET BASESTACK   ; INITIALIZE SP
          MOV     AX,DATA               ; INITIALIZE
          MOV     DS,AX                 ;   DS
          .
          .
          .
          CALL    CI                    ; INPUT CHARACTER FROM CONSOLE
          AND     AL,7FH                ; STRIP OFF PARITY BIT
          MOV     CHAR,AL               ; SAVE CHARACTER
          .
          .
          .
CODE      ENDS
          END     INIT
```

Example 3

```
ASM86 CI Intersegment Call Example

          EXTRN   CI:FAR
          ASSUME  DS:MYDATA,SS:STACK,CS:MYCODE

MYDATA    SEGMENT 'DATA'
CHAR      DB      ?
MYDATA    ENDS

STACK     SEGMENT STACK 'STACK'
          DW      16 DUP ?
BASESTACK LABEL   WORD
STACK     ENDS

MYCODE    SEGMENT 'CODE'
INIT:
          MOV     AX,STACK              ; INITIALIZE
          MOV     SS,AX                 ;   SS
          MOV     SP,OFFSET BASESTACK   ; INITIALIZE SP
          MOV     AX,MYDATA             ; INITIALIZE
          MOV     DS,AX                 ;   DS
          .
          .
          .
          CALL    CI                    ; INPUT CHARACTER FROM CONSOLE
          AND     AL,7FH                ; STRIP OFF PARITY BIT
          MOV     CHAR,AL               ; SAVE CHARACTER
          .
          .
          .
MYCODE    ENDS
          END     INIT
```

4-9. Console Output (CO) Routine

This routine transmits an 8-bit character, passed from the caller on the stack in the low-order byte, to the console device. The AX, CX, and DX registers and the CPU conditions codes are affected by this operation.

Before the character is output, the routine checks to see if a Control-S has been input. If so, it waits until a Control-Q is input before outputting the character. Thus, if a Control-S is entered at the console when output is pending, the output is temporarily stopped. Entering a Control-Q will resume output.

Example 1

```
PL/M-86 CO Call Example

CO: PROCEDURE(X) EXTERNAL;
  DECLARE X BYTE;
  END CO;

DECLARE CHAR BYTE;
    .
    .
    .
CALL CO(CHAR);     /* OUTPUT CHARACTER */
    .
    .
    .
```

Example 2

```
ASM86 CO Intrasegment Call Example

          ASSUME  DS:DATA,SS:STACK,CS:CODE

DATA      SEGMENT PUBLIC 'DATA'
CHAR      DB      ?
DATA      ENDS

STACK     SEGMENT STACK 'STACK'
          DW      15 DUP ?
BASESTACK LABEL   WORD
STACK     ENDS

CODE      SEGMENT PUBLIC 'CODE'
          EXTRN   CO:NEAR
INIT:
          MOV     AX,STACK              ; INITIALIZE
          MOV     SS,AX                 ;   SS
          MOV     SP,OFFSET BASESTACK   ; INITIALIZE SP
          MOV     AX,DATA               ; INITIALIZE
          MOV     DS,AX                 ;   DS
          .
          .
          .
          MOV     AL,CHAR               ; LOAD CHARACTER INTO AL
          PUSH    AX                    ; PUSH CHARACTER ONTO STACK
          CALL    CO                    ; OUTPUT CHARACTER TO CONSOLE
          .
          .
          .
CODE      ENDS
          END     INIT
```

Example 3

```
ASM86 CO Intersegment Call Example

          EXTRN   CO:FAR
          ASSUME  DS:MYDATA,SS:STACK,CS:MYCODE

MYDATA    SEGMENT 'DATA'
CHAR      DB      ?
MYDATA    ENDS

STACK     SEGMENT STACK 'STACK'
          DW      16 DUP ?
BASESTACK LABEL   WORD
STACK     ENDS

MYCODE    SEGMENT 'CODE'
INIT:
          MOV     AX,STACK              ; INITIALIZE
          MOV     SS,AX                 ;   SS
          MOV     SP,OFFSET BASESTACK   ; INITIALIZE SP
          MOV     AX,MYDATA             ; INITIALIZE
          MOV     DS,AX                 ;   DS
          .
          .
          .
          MOV     AL,CHAR               ; LOAD CHARACTER INTO AL
          PUSH    AX                    ; PUSH CHARACTER ONTO STACK
          CALL    CO                    ; OUTPUT CHARACTER TO CONSOLE
          .
          .
          .
MYCODE    ENDS
          END     INIT
```

APPENDIX A
iSBC 86/12 MONITOR
(PL/M-86 COMPILER SOURCE LISTING)

```
ISIS-II PL/M-86 V1.0 COMPILATION OF MODULE MONITOR
OBJECT MODULE PLACED IN :F1:SBCMON.OBJ
COMPILER INVOKED BY:  :F2:PLM86 :F1:SBCMON.PLM LARGE OPTIMIZE(2) PRINT(:F3:SBCMON.LST) &
                      PAGEWIDTH(95)


$TITLE('ISBC 86/12 MONITOR')
$NOINTVECTOR

 /* ***********************************************************************
    ***********************************************************************

        ISBC 86/12 MONITOR, V1.2
        18 JULY 1978

        (C) 1978 INTEL CORPORATION. ALL RIGHTS RESERVED. NO PART
     OF THIS PROGRAM OR PUBLICATION MAY BE REPRODUCED, TRANSMITTED,
     TRANSCRIBED, STORED IN A RETRIEVAL SYSTEM, OR TRANSLATED INTO ANY
     LANGUAGE, IN ANY FORM OR BY ANY MEANS, ELECTRONIC, MECHANICAL,
     MAGNETIC, OPTICAL, CHEMICAL, MANUAL OR OTHERWISE, WITHOUT THE PRIOR
     WRITTEN PERMISSION OF INTEL CORPORATION, 3065 BOWERS AVENUE, SANTA
     CLARA, CALIFORNIA, 95051.

     ***********************************************************************
     ***********************************************************************

     ABSTRACT
     ========
     THIS PROGRAM IS THE ROM BASED MONITOR FOR THE ISBC 86/12.  IT PROVIDES
     THE USER WITH A MODERATE LEVEL OF CAPABILITY TO EXAMINE/MODIFY
     MEMORY AND REGISTERS, CONTROL PROGRAM EXECUTION, AND LOAD/SAVE
     PROGRAMS.

     ENVIRONMENT
     ===========
     THE SBC MONITOR COMMUNICATES WITH THE USER VIA AN INTERACTIVE
     TERMINAL (TTY,CRT) ATTACHED TO THE SERIAL PORT.

     PROGRAM ORGANIZATION
     ====================
     THE PROGRAM IS DIVIDED INTO 1 DATA AND 2 CODE MODULES:
        1. DATA DECLARATION MODULE.     GLOBAL DATA DECLARATIONS.
        2. COMMON ROUTINES.             LOWER LEVEL PROCEDURES
        3. COMMAND MODULE.              INDIVIDUAL COMMANDS AND OUTER BLOCK

     CALLING GRAPH
     =============
        >>COMMAND DISPATCH MODULE (OUTER BLOCK)
             INDIVIDUAL COMMAND PROCEDURES
                COMMON ROUTINES

     GLOBAL DATA STRUCTURES
     ======================
     THE MONITOR MAINTAINS THE USER'S MACHINE STATE (REGISTERS) IN A
     WORD ARRAY. THE REGISTERS ARE SAVED FROM THE USER'S STACK
     AS PUSHED BY PLM86 INTERRUPT PROCEDURE.
```

```
                POINTERS TO THE 2**20 ADDRESS SPACE ARE IMPLEMENTED WITH
                POINTER STRUCTURES ALLOCATED AS 2 WORD STRUCTURES.

             */

1            MONITOR:DO;                    /* BEGINNING OF MODULE */

             /* ***************************************************************
                ***************************************************************

                              GLOBAL DATA DECLARATIONS MODULE
                              ===============================
                ABSTRACT
                ========
                THIS MODULE CONTAINS ALL THE GLOBAL DATA DECLARATIONS AND
                LITERALS (EQUATES).

                MODULE ORGAINIZATION
                ====================
                THE MODULE IS DIVIDED INTO 5 SECTIONS:
                   1. UTILITY SECTION              GLOBAL FLAGS,VARIBLES,EQUATES
                   2. I/O SECTION                  I/O PORTS,MASKS,AND SPECIAL CHARS
                   3. MEMORY ARGUMENTS SECTIONS    STRUCTURES FOR POINTERS
                   4. REGISTER SECTION             USER REGISTER SAVE AREA
             */

                /************************
                *   UTILITY SECTION    *
                ************************/

             DECLARE
                    INT$VECTOR(40)     POINTER;          /* INTERRUPT VECTORS */
             DECLARE
                    MONITOR$STACKPTR   WORD,
                    MONITOR$STACKBASE  WORD;
             DECLARE
                    INT3$PTR           POINTER;          /* INTERRUPT3$ENTRY ADDRESS */
             DECLARE
                    COPYRIGHT(*) BYTE DATA ('(C) 1978 INTEL CORP');
             DECLARE
                    BRK1$FLAG       BYTE,                /* TRUE IF BREAK SET */
                    BRK1$SAVE       BYTE,                /* INST BREAK SAVE */
                    BRK2$FLAG       BYTE,                /* TRUE IF BREAK 2 SET */
                    BRK2$SAVE       BYTE,                /* INST BREAK 2 SAVE */
                    CHAR            BYTE,                /* ONE CHAR LOOK AHEAD */
                    CHECK$SUM       BYTE,                /* PAPER TAPE CHECKSUM */
                    I               BYTE,                /* INDEX */
                    J               BYTE,                /* INDEX */
                    II              WORD,                /* INDEX */
                    JJ              WORD,                /* INDEX */
                    END$OFF         WORD,                /* END OFFSET ADDRESS */
                    WORD$MODE       BYTE,                /* WORD MODE FLAG */
                    LAST$COMMAND    BYTE,                /* LAST COMMAND SAVE */
                    MODE            BYTE,               /* R, W, L & T MODE */
                    SAVE$MODE       BYTE,               /* SAVE MODE */
                    SWITCH$BAUD     BYTE,               /* BAUD RATE SWITCH FLAG */
                    MODE$8086       BYTE,               /* 8086 FILE FORMAT */
```

```
                  BRF             WORD;                   /* BAUD RATE FACTOR */

  7   1       DECLARE
                  TRUE            LITERALLY '0FFH',
                  FALSE           LITERALLY '000H',
                  BREAK$INST      LITERALLY '0CCH',       /* BREAKPOINT INST */
                  STEP$TRAP       LITERALLY '0100H',      /* SS TRAP FLAG MASK */
                  USER$INIT$SP    LITERALLY '1C0H',       /* USER STACK INITIAL */
                  GO$COMMAND      LITERALLY '2',          /* GO COMMAND CODE */
                  SS$COMMAND      LITERALLY '3',          /* SINGLE STEP CODE */
                  STANDARD$LEN    LITERALLY '16',         /* PAPER TAPE DATA REC LEN */
                  MAX$DELAY       LITERALLY '60000',      /* DELAY FOR READ CHAR */
                  TAPE            LITERALLY '1H',         /* TAPE MODE */
                  SERIAL          LITERALLY '2H',         /* SERIAL MODE */
                  PARALLEL        LITERALLY '4H',         /* PARALLEL MODE */
                  ASCR            LITERALLY  '0DH',       /* CARRIAGE RETURN */
                  ASLF            LITERALLY  '0AH',       /* LINE FEED       */
                  ASBL            LITERALLY  '20H';       /* BLANK OR SPACE  */
  8   1       DECLARE
                  SIO$BREAK1$MSG(*) BYTE DATA  ('BR1 ',0),
                  SIO$BREAK2$MSG(*) BYTE DATA  ('BR2 ',0),
                  SIO$SIGNON(*)     BYTE DATA  (0DH,0AH,'ISBC 86/12 MONITOR, V1.2',0),
                  ASCII(*)          BYTE DATA  ('0123456789ABCDEF'),
                  SIO$CMND(*)       BYTE DATA  ('SXGNMDCFHIORWLT'),
                  BR$CHAR(*)        BYTE DATA  (55H,66H,78H);

                /*********************************
                *   I/O DECLARATIONS SECTION    *
                *********************************/

  9   1       DECLARE                                     /* 8251A USART */
                  SIO$STAT$PORT   LITERALLY '0DAH',       /* STATUS PORT */
                  SIO$DATA$PORT   LITERALLY '0D8H',       /* DATA PORT */
                  SIO$RESET       LITERALLY '40H',        /* RESET USART */
                  SIO$CRT$MODE    LITERALLY '4EH',        /* CRT MODE */
                  SIO$TTY$MODE    LITERALLY '0CFH',       /* TTY MODE */
                  SIO$DTR$ON      LITERALLY '27H',        /* RTS,RXE,DTR,TXE */
                  SIO$CRT$CMD     LITERALLY '37H',        /* RTS,ER,RXE,DTR,TXE */
                  SIO$TTY$CMD     LITERALLY '35H',        /* RTS,ER,RXE,TXE */
                  SIO$DTR$OFF     LITERALLY '25H',        /* RTS,RXE,TXE */
                  SIO$RXRDY       LITERALLY '02H',        /* RECEIVER READY */
                  SIO$TXE         LITERALLY '04H',        /* TRANSMITTER EMPTY */
                  SIO$TXRDY       LITERALLY '01H',        /* TRANSMITTER READY */
                  PARITY$MASK     LITERALLY '7FH';        /* MASK OFF PARITY BIT */

 10   1       DECLARE                                     /* 8253 INTERVAL TIMER */
                  IT$CONTROL$PORT LITERALLY '0D6H',       /* CONTROL PORT */
                  IT$CTR2$PORT    LITERALLY '0D4H',       /* COUNTER 2 PORT */
                  IT$C2M3         LITERALLY '0B6H',       /* COUNTER 2, MODE 3 */
                  B9600           LITERALLY '0008H',      /* TIMER VALUE FOR 9600 BAUD */
                  B1200           LITERALLY '0040H',     /* TIMER VALUE FOR 1200 BAUD */
                  B600            LITERALLY '0080H',      /* TIMER VALUE FOR 600 BAUD */
                  B110            LITERALLY '00AFH';     /* TIMER VALUE FOR 110 BAUD */

                DECLARE                                   /* 8259A INTERRUPT CONTROLLER */
                  IC$PORTA        LITERALLY '0C0H',       /* PORT A */
                  IC$PORTB        LITERALLY '0C2H',       /* PORT B */
```

```
                IC$ICW1         LITERALLY '17H',        /* INIT COMMAND WORD 1 */
                IC$ICW2         LITERALLY '20H',        /* INIT COMMAND WORD 2 */
                IC$ICW4         LITERALLY '1DH',        /* INIT COMMAND WORD 4 */
                IC$MASK         LITERALLY '00H',        /* INTERRUPT MASK */
                IC$OCW3         LITERALLY '0BH',        /* READ INTERRUPT LEVEL */
                IC$EOI          LITERALLY '20H';        /* END OF INTERRUPT CMD */

12  1       DECLARE                                   /* 8255A PERIPHERAL INTERFACE */
                PI$PORTA        LITERALLY '0C8H',       /* PORT A (OUTPUT) */
                PI$PORTB        LITERALLY '0CAH',       /* PORT B (INPUT) */
                PI$PORTC$STAT   LITERALLY '0CCH',       /* PORT C STATUS */
                PI$PORTC$CTL    LITERALLY '0CEH',       /* PORT C CONTROL */
                PI$M2M1         LITERALLY '0C6H',       /* A-MODE 1, B-MODE 2 */
                PI$OBF          LITERALLY '080H',       /* OUTPUT BUFFER READY */
                PI$IBF          LITERALLY '02H';        /* INPUT BUFFER READY */

            /*******************************
            *   MEMORY ARGUMENT SECTION    *
            *******************************/

13  1       DECLARE
                MEMORY$ARG1$PTR  POINTER,               /* ARGUMENT 1 */
                ARG1 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$ARG1$PTR),
                MEMORY$ARG1 BASED MEMORY$ARG1$PTR BYTE,
                MEMORY$WORD$ARG1 BASED MEMORY$ARG1$PTR WORD,

                MEMORY$ARG2$PTR POINTER,                /* ARGUMENT 2 */
                ARG2 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$ARG2$PTR),
                MEMORY$ARG2 BASED MEMORY$ARG2$PTR BYTE,

                MEMORY$ARG3$PTR POINTER,                /* ARGUMENT 3 */
                ARG3 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$ARG3$PTR),
                MEMORY$ARG3 BASED MEMORY$ARG3$PTR BYTE,

                MEMORY$BRK1$PTR POINTER,                /* BREAKPOINT 1 */
                BRK1 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$BRK1$PTR),
                MEMORY$BRK1 BASED MEMORY$BRK1$PTR BYTE,

                MEMORY$BRK2$PTR POINTER,                /* BREAKPOINT 2 */
                BRK2 STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$BRK2$PTR),
                MEMORY$BRK2 BASED MEMORY$BRK2$PTR BYTE,

                MEMORY$CSIP$PTR POINTER,                /* CS & IP WORD */
                CSIP STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$CSIP$PTR),
                MEMORY$CSIP BASED MEMORY$CSIP$PTR BYTE,

                MEMORY$USERSTACK$PTR POINTER,
                USERSTACK STRUCTURE (OFF WORD, SEG WORD)
                   AT (@MEMORY$USERSTACK$PTR),
                MEMORY$USERSTACK BASED MEMORY$USERSTACK$PTR WORD;
```

```
                /************************
                *   REGISTER SECTION   *
                ************************/

 14   1       DECLARE
                   REG(*)          BYTE DATA              /* REGISTER NAMES */
                      ('AXBXCXDXSPBPSIDICSDSSSESIPFL'),
                   REG$INDEX       WORD,
                   REG$SAV(14)     WORD,                  /* USER'S SAVED REGS */
                   REG$ORD(*)      BYTE DATA
                      (7,6,1,3,2,0,9,11,12,8,13),
                   SP              LITERALLY 'REG$SAV( 4)',
                   BP              LITERALLY 'REG$SAV( 5)',
                   CS              LITERALLY 'REG$SAV( 8)',
                   DS              LITERALLY 'REG$SAV( 9)',
                   SS              LITERALLY 'REG$SAV(10)',
                   ES              LITERALLY 'REG$SAV(11)',
                   IP              LITERALLY 'REG$SAV(12)',
                   FL              LITERALLY 'REG$SAV(13)';


               /* ************************************************************************
                  ************************************************************************

                                  COMMON PROCEDURES
                                  =================
                  ABSTRACT
                  ========
                  THIS MODULE CONTAINS THOSE LOWER LEVEL PROCEDURES CALLED BY HIGHER
                  LEVEL ROUTINES.

                  MODULE ORGANIZATION
                  ===================
                  THIS MODULE IS DIVIDED INTO 4 SECTIONS AS FOLLOWS:
                     1. BASIC I/O SECTION
                         SIO$CHAR$RDY          INPUT CHARACTER READY
                         SIO$CHECK$CONTROL$CHAR  CHECK FOR CONTROL CHARACTER
                         SIO$OUT$CHAR          OUTPUT CHARACTER
                         SIO$GET$CHAR          INPUT A CHARACTER
                         SIO$OUT$BYTE          OUTPUT A BYTE IN HEX
                         SIO$OUT$BYTE$PTR      OUTPUT BYTE AT POINTER
                         SIO$OUT$WORD          OUTPUT A WORD IN HEX
                         SIO$OUT$BLANK         OUTPUT A SINGLE BLANK
                         SIO$OUT$STRING        OUTPUT A STRING
                         SIO$OUT$HEADER        OUTPUT A HEX FILE HEADER
                     2. UTILITY ROUTINES SECTION
                         SIO$VALID$HEX         TEST FOR VALID HEX CHAR
                         SIO$HEX               CONVERT TO HEX FROM ASCII
                         SIO$VALID$REG$FIRST   TEST FOR VALID REGISTER FIRST CHAR
                         SIO$VALID$REG         TEST FOR VALID REGISTER NAME
                         SIO$CRLF              OUTPUTS A CR AND LF
                         SIO$TEST$WORD$MODE    TEST FOR A 'W' IN COMMAND
                         SIO$SCAN$BLANK        SCANS FOR OPTIONAL BLANK
                         SIO$SECOND$DELAY      DELAY ONE SECOND
                         SIO$MS$DELAY          DELAY N MS.
                     3. ARGUMENT EXPRESSION EVALUATION SECTION
```

```
                        SIO$GET$WORD          GET AN WORD EXPRESSION
                        SIO$GET$ADDR          GET AN ADDRESS EXPRESSION
                        SIO$UPDATE$IP         OPTIONAL UPDATE OF CS:IP
                    4. DEVICE INITIALIZATION SECTION
                        SIO$RESET$USART       RESET 8251A
                        SIO$INIT$MODE         INITIALIZE FOR LOAD/TRANSFER
                    5. PAPER TAPE, SERIAL, PARALLEL READ SECTION
                        SIO$READ$CHAR         READ CHAR FROM TTY READER
                        SIO$READ$BYTE         READ A BYTE
                        SIO$READ$WORD         READ A WORD
                        SIO$WRITE$HEX$FILE    OUTPUT HEX FILE
                        SIO$READ$HEX$FILE     INPUT HEX FILE
                    6. INTERRUPT AND RESTORE/EXECUTE ROUTINES
                        SAVE$REGISTERS        SAVES USER REGISTERS
                        RESTORE$EXECUTE       RESTORE MACHINE STATE AND EXEC
                        INTERRUPT1$ENTRY      INTERRUPT ROUTINE FOR SINGLE STEP
                        INTERRUPT3$ENTRY      INTERRUPT ROUTINE FOR GO
                        INTERRUPT32$ENTRY     INTERRUPT ROUTINE FOR 8259A
                        INIT$INT$VECTOR       INITIALIZES INTERRRUPT VECTORS
                */

                   /************************
                   *   BASIC I/O SECTION   *
                   ************************/

 15   1        SIO$CHAR$RDY:
                   /* TESTS FOR INPUT CHARACTER PENDING BY READING THE STATUS PORT
                      AND MASKING WITH SIO$RXRDY(READ DATA READY).  RETURNS TRUE IF
                      NOT EMPTY(CHAR PENDING) AND FALSE IF NO CHAR PENDING */
                   PROCEDURE BYTE;
 16   2                 IF (INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0 THEN RETURN FALSE;
 18   2                 RETURN TRUE;
 19   2              END;

 20   1        SIO$CHECK$CONTROL$CHAR:
                   /* THIS ROUTINE CHECKS IF A CONTROL CHARACTER HAS BEEN INPUT TO
                      THE SERIAL PORT.  AFTER A CONTROL-S IT WAITS FOR A CONTROL-Q BEFOR
                      RETURNING TO THE CALLER.  A CONTROL-C CAUSES A JUMP TO THE ERROR
                      ROUTINE. */
                   PROCEDURE;
 21   2                 CHAR = INPUT(SIO$DATA$PORT) AND 07FH;
 22   2                 IF CHAR=13H THEN                /* CONTROL-S ? */
 23   2                    DO WHILE CHAR<>11H;          /* CONTROL-Q   */
 24   3                       IF SIO$CHAR$RDY THEN
 25   3                          DO;
 26   4                          CHAR = INPUT(SIO$DATA$PORT) AND 07FH;
 27   4                          IF CHAR=03H THEN GOTO ERROR;
 29   4                          END;
 30   3                    END;
 31   2                 ELSE IF CHAR = 03H THEN GOTO ERROR;
                   END SIO$CHECK$CONTROL$CHAR;

 34   1        SIO$OUT$CHAR:
                   /* THIS ROUTINE OUTPUTS THE INPUT PARAMETER TO THE USART OUTPUT
                      PORT WHEN USART IS READY FOR OUTPUT (XMIT BUFFER EMPTY). */
                   PROCEDURE(C);
 35   2                 DECLARE C BYTE;
```

```
 36   2              IF (MODE AND SERIAL) <> 0 THEN
 37   2                 DO;
 38   3        LOOP:    IF SIO$CHAR$RDY THEN CALL SIO$CHECK$CONTROL$CHAR;
 40   3                 IF (INPUT(SIO$STAT$PORT) AND SIO$TXRDY) = 0 THEN GOTO LOOP;
 42   3                 OUTPUT(SIO$DATA$PORT) = C;
 43   3                 END;
                   ELSE
 44   2                 DO;
 45   3        LOOP1:   IF SIO$CHAR$RDY THEN CALL SIO$CHECK$CONTROL$CHAR;
 47   3                 IF (INPUT(PI$PORTC$STAT) AND PI$OBF) = 0 THEN GOTO LOOP1;
 49   3                 OUTPUT(PI$PORTA) = C;
 50   3                 END;
 51   2              RETURN;
 52   2           END;

 53   1        SIO$GET$CHAR:
                   /* THIS ROUTINE INPUTS A CHARACTER FROM THE INPUT PORT AND RETURNS
                      WITH IT IN THE GLOBAL 'CHAR'. THE CHARACTER IS ECHOED TO THE
                      OUTPUT PORT IF PRINTABLE. */
                   PROCEDURE;
 54   2                 DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0;END;
 56   2                 CHAR = INPUT(SIO$DATA$PORT) AND 07FH;
 57   2                 IF CHAR>=ASBL THEN CALL SIO$OUT$CHAR(CHAR);
 59   2              END;

 60   1        SIO$OUT$BYTE:
                   /* THIS ROUTINE OUTPUTS THE SINGLE INPUT PARAMETER TO THE USART
                      IN ASCII HEXADECIMAL FORMAT. */
                   PROCEDURE(B);
 61   2                 DECLARE B BYTE;
 62   2                 CALL SIO$OUT$CHAR(ASCII(SHR(B,4) AND 0FH));
 63   2                 CALL SIO$OUT$CHAR(ASCII(B AND 0FH));
 64   2                 CHECK$SUM = CHECK$SUM - B;
 65   2              END;

 66   1        SIO$OUT$BYTE$PTR:
                   /* THIS ROUTINE OUTPUTS THE BYTE BASED ON THE INPUT PARAMETER TO THE USAI
                      IN ASCII HEXADECIMAL FORMAT. */
                   PROCEDURE (B$PTR);
 67   2                 DECLARE B$PTR POINTER, B BASED B$PTR BYTE, X BYTE;
 68   2                 X = B;
 69   2                 CALL SIO$OUT$BYTE(X);
 70   2              END SIO$OUT$BYTE$PTR;

 71   1        SIO$OUT$WORD:
                   /* THIS ROUTINE OUTPUTS THE INPUT PARAMETER AS 4 ASCII HEXADECIMAL
                      CHARACTERS TO THE USART OUTPUT PORT. */
                   PROCEDURE(W);
 72   2                 DECLARE W WORD;
 73   2                 CALL SIO$OUT$BYTE(HIGH(W));
 74   2                 CALL SIO$OUT$BYTE(LOW(W));
 75   2              END;

 76   1        SIO$OUT$BLANK:
                   /* THIS ROUTINE OUTPUTS ONE BLANK. */
                   PROCEDURE;
 77   2                 CALL SIO$OUT$CHAR(ASBL);
```

```
 78   2          END;

 79   1       SIO$OUT$STRING:
                 /* OUTPUTS A STRING POINTED TO BY THE FIRST PARM. */
                 PROCEDURE(PTR);
 80   2             DECLARE PTR POINTER, STR BASED PTR (1) BYTE;
 81   2             I = 0;
 82   2             DO WHILE STR(I)<>0;
 83   3                CALL SIO$OUT$CHAR(STR(I));
 84   3                I = I + 1;
 85   3             END;
 86   2          END;

 87   1       SIO$OUT$HEADER:
                 /* THIS ROUTINE OUTPUTS THE HEX FILE HEADER CONSISTING OF ':'
                    FOLLOWED BY THE RECORD LENGTH, LOAD ADDRESS, AND THE RECORD TYPE.
                    IT INITIALIZES THE CHECKSUM TO ZERO. */
                 PROCEDURE(LENGTH,LOAD$ADDR,REC$TYPE);
 88   2             DECLARE (LENGTH,REC$TYPE) BYTE, LOAD$ADDR WORD;
 89   2             CALL SIO$OUT$CHAR(':');
 90   2             CHECK$SUM = 0;
 91   2             CALL SIO$OUT$BYTE(LENGTH);
 92   2             CALL SIO$OUT$WORD(LOAD$ADDR);
 93   2             CALL SIO$OUT$BYTE(REC$TYPE);
 94   2          END;


                  /*********************************
                  *   UTILITY ROUTINES SECTION    *
                  *********************************/

 95   1       SIO$VALID$HEX:
                 /* THIS ROUTINE TESTS IF THE INPUT PARM IS A VALID ASCII HEX DIGIT
                    AND RETURNS TRUE AS THE VALUE OF THE PROCEDURE IF SO AND FALSE
                    IF NOT. */
                 PROCEDURE (H) BYTE;
 96   2             DECLARE H BYTE;
 97   2             DO I=0 TO LAST(ASCII);
 98   3                IF H=ASCII(I) THEN RETURN TRUE;
100   3             END;
101   2             RETURN FALSE;
102   2          END;

103   1       SIO$HEX:
                 /* THIS ROUTINE CONVERTS THE INPUT PARM FROM ASCII TO ITS BINARY
                    EQUIVALENT AND RETURNS IT AS THE VALUE OF THE PROCEDURE. NO CHECK
                    IS MADE FOR INPUT VALIDITY. */
                 PROCEDURE(C) WORD;
104   2             DECLARE C BYTE;
105   2             IF C<='9' THEN RETURN DOUBLE(C-30H);
107   2             ELSE RETURN DOUBLE(C-37H);
108   2          END;

109   1       SIO$VALID$REG$FIRST:
                 /* THIS ROUTINE CHECKS IF 'CHAR' IS A VALID FIRST LETTER OF A REGISTER
                    NAME AND RETURNS TRUE AS THE VALUE OF THE PROCEDURE IF SO. */
                 PROCEDURE BYTE;
```

```
110   2             DO I=0 TO 26 BY 2;
111   3                IF CHAR=REG(I) THEN RETURN TRUE;
113   3             END;
114   2             RETURN FALSE;
115   2          END;

116   1       SIO$VALID$REG:
                  /* THIS ROUTINE CHECKS IF THE TWO INPUT PARMS TAKEN TOGETHER FORM
                     A VALID REGISTER NAME.  IT SEARCHES THE REGISTER TABLE AND IF A
                     MATCH IS FOUND, THE GLOBAL 'REG$INDEX' IS SET TO THE INDEX OF THE
                     VALID REGISTER AND THE PROCEDURE RETURNS TRUE.  IF NO MATCH THE
                     PROCEDURE RETURNS FALSE AND REG$INDEX IS UNDEFINED. */
                  PROCEDURE (C1,C2) BYTE;
117   2             DECLARE (C1,C2) BYTE;
118   2             DO REG$INDEX=0 TO 13;
119   3                IF C1=REG(REG$INDEX*2) AND C2=REG(REG$INDEX*2+1) THEN
120   3                   RETURN TRUE;
121   3             END;
122   2             RETURN FALSE;
123   2          END;

124   1       SIO$CRLF:
                  /* THIS ROUTINE OUTPUTS A CR AND LF TO THE OUTPUT PORT. */
                  PROCEDURE;
125   2             CALL SIO$OUT$CHAR(ASCR);
126   2             CALL SIO$OUT$CHAR(ASLF);
127   2          END;

128   1       SIO$TEST$WORD$MODE:
                  /* THIS PROCEDURE TESTS FOR A 'W' FOLLOWING THE COMMAND AND IF SO
                     SETS THE FLAG 'WORD$MODE TO TRUE OR FALSE OTHERWISE. SCANS OFF
                     OPTIONAL BLANK FOLLOWING COMMAND. */
                  PROCEDURE;
129   2             WORD$MODE = FALSE;
130   2             CALL SIO$GET$CHAR;
131   2             IF CHAR='W' THEN
132   2                DO;
133   3                WORD$MODE = TRUE;
134   3                CALL SIO$GET$CHAR;
135   3                END;
136   2             IF CHAR=ASBL THEN
137   2                CALL SIO$GET$CHAR;
138   2          END;

139   1       SIO$SCAN$BLANK:
                  /* THIS ROUTINE IS CALLED AFTER A COMMAND LETTER TO SCAN OFF THE
                     OPTIONAL BLANK. */
                  PROCEDURE;
140   2             CALL SIO$GET$CHAR;
141   2             IF CHAR=ASBL THEN
142   2                CALL SIO$GET$CHAR;
143   2          END;


144   1       SIO$SECOND$DELAY:
                  /* THIS ROUTINE CAUSES A DELAY OF APPROXIMATELY 1 SECOND. */
                  PROCEDURE;
```

```
145   2             DECLARE II WORD;
146   2             DO II = 1 TO 0F000H; END;
148   2          END SIO$SECOND$DELAY;


149   1       SIO$MS$DELAY:
                /* THIS ROUTINE CAUSES A DELAY OF 1 OR MORE MILLESECONDS; THE NUMBER
                   IS PASSED BY THE CALLER.  THE DELAY IS APPROXIMATE. */
                PROCEDURE (N);
150   2             DECLARE (N,I,J) BYTE;
151   2             DO I = 1 TO N;
152   3                DO J = 1 TO 55; END;
154   3             END;
155   2          END SIO$MS$DELAY;


                /*******************************************
                *   ARGUMENT EXPRESSION EVALUATOR SECTION   *
                *******************************************/

156   1       SIO$GET$WORD:
                /* THIS ROUTINE READS CHARS FROM THE INPUT PORT AND EVALUATES
                   AN EXPRESSION CONSISTING OF '+-' AND OPERANDS OF HEX NUMBERS
                   AND REGISTER NAMES. */
                PROCEDURE WORD;
157   2             DECLARE (SAVE,W) WORD, (OPER,T) BYTE;
158   2             OPER = '+';
159   2             W = 0;
160   2             DO WHILE TRUE;
161   3                T = CHAR;
162   3                SAVE = 0;
163   3                IF SIO$VALID$REG$FIRST THEN
164   3                   DO;
165   4                   CALL SIO$GET$CHAR;
166   4                   IF SIO$VALID$REG(T,CHAR) THEN
167   4                      DO;
168   5                      SAVE = REG$SAV(REG$INDEX);
169   5                      CALL SIO$GET$CHAR;
170   5                      GOTO EVAL;
171   5                      END;
                           ELSE
172   4                      SAVE = SIO$HEX(T);
173   4                   END;
174   3                IF NOT(SIO$VALID$HEX(T)) THEN  GOTO ERROR;
176   3                DO WHILE SIO$VALID$HEX(CHAR);
177   4                   SAVE = SHL(SAVE,4) + SIO$HEX(CHAR);
178   4                   CALL SIO$GET$CHAR;
179   4                END;
180   3       EVAL:    IF OPER='+' THEN
181   3                   W = W + SAVE;
                         ELSE
182   3                   W = W - SAVE;
183   3                IF CHAR=ASCR OR CHAR=':' OR CHAR=',' THEN
184   3                   RETURN W;
185   3                IF CHAR='+' OR CHAR='-' THEN
186   3                   OPER = CHAR;
                         ELSE
```

```
187   3                  GOTO ERROR;
188   3               CALL SIO$GET$CHAR;
189   3             END;
190   2          END;

191   1       SIO$GET$ADDR:
                  /* THIS ROUTINE ACCEPTS A VALID ADDRESS EXPRESSION CONSISTING
                     OF AN OPTIONAL <SEG>: AND AN DISPLACEMENT. */
                  PROCEDURE(PTR,DEFAULT$BASE);
192   2              DECLARE PTR POINTER, DEFAULT$BASE WORD,
                        ARG BASED PTR STRUCTURE (OFF WORD, SEG WORD);
193   2              ARG.SEG = DEFAULT$BASE;
194   2              ARG.OFF = SIO$GET$WORD;
195   2              IF CHAR=':' THEN
196   2                 DO;
197   3                 CALL SIO$GET$CHAR;
198   3                 ARG.SEG = ARG.OFF;
199   3                 ARG.OFF = SIO$GET$WORD;
200   3                 IF CHAR=':' THEN  GOTO ERROR;
202   3                 END;
203   2          END;

204   1       SIO$UPDATE$IP:
                  /* THIS PROCEDURE IS CALLED BY SINGLE STEP AND GO TO OUTPUT THE CURRENT
                     IP AND INSTRUCTION BYTE AND OPEN THE IP FOR INPUT. */
                  PROCEDURE;
205   2             CALL SIO$OUT$BLANK;
206   2             CALL SIO$OUT$WORD(IP);
207   2             CSIP.SEG = CS;
208   2             CSIP.OFF = IP;
209   2             CALL SIO$OUT$CHAR('-');
210   2             CALL SIO$OUT$BLANK;
211   2             CALL SIO$OUT$BYTE$PTR(MEMORY$CSIP$PTR);
212   2             CALL SIO$OUT$BLANK;
213   2             CALL SIO$GET$CHAR;
214   2             IF CHAR<>',' AND CHAR<>ASCR THEN CALL SIO$GET$ADDR(@CSIP,CS);
216   2          END;



                  /*******************************************
                  *    DEVICE INITIALIZATION SECTION          *
                  *******************************************/


217   1       SIO$RESET$USART:
                    /* THIS PROCEDURE RESETS THE 8251A USART */
                    PROCEDURE;
218   2                 OUTPUT(SIO$STAT$PORT) = 0H;
219   2                 CHAR = 0;  /* DELAY */
220   2                 OUTPUT(SIO$STAT$PORT) = 0H;
221   2                 CHAR = 0;  /* DELAY */
222   2                 OUTPUT(SIO$STAT$PORT) = 0H;
223   2                 CHAR = 0;  /* DELAY */
224   2                 OUTPUT(SIO$STAT$PORT) = SIO$RESET;
225   2             END SIO$RESET$USART;
```

```
226   1       SIO$INIT$MODE:
                 /* INITIALIZES THE INTERFACE FOR THE LOAD AND TRANSFER COMMANDS */
                 PROCEDURE;
227   2             MODE = SAVE$MODE;
228   2             IF MODE = PARALLEL THEN OUTPUT(PI$PORTC$CTL) = PI$M2M1;
                    ELSE
230   2                DO;
231   3                IF (MODE AND TAPE) = 0 THEN
232   3                   IF BRF = B600 THEN
233   3                      DO;
                              /* MUST BE ON-LINE TO INTELLEC SERIES II WITH
                                 INTEGRATED CRT.  JACK-UP BAUD RATE. */
234   4                      CALL SIO$MS$DELAY(200);
235   4                      CALL SIO$RESET$USART;
236   4                      OUTPUT(SIO$STAT$PORT) = SIO$CRT$MODE;
237   4                      OUTPUT(IT$CONTROL$PORT) = IT$C2M3;
238   4                      OUTPUT(SIO$STAT$PORT) = SIO$CRT$CMD;
239   4                      OUTPUT(IT$CTR2$PORT) = LOW(B9600);
240   4                      OUTPUT(IT$CTR2$PORT) = HIGH(B9600);
241   4                      SWITCH$BAUD = TRUE;
242   4                      END;
243   3                END;
244   2          END SIO$INIT$MODE;


                 /************************************************
                 *   PAPER TAPE, SERIAL, PARALLEL READ SECTION   *
                 ************************************************/

245   1       SIO$READ$CHAR:
                 /* THIS PROCEDURE READS A BYTE FROM THE TTY PAPER TAPE READER,
                    THE SERIAL INTERFACE TO AN INTELLEC, OR THE PARALLEL
                    INTERFACE TO AN INTELLEC, DEPENDING UPON THE SETTING OF
                    MODE. */
                 PROCEDURE BYTE;
246   2             DECLARE II WORD;
247   2             IF (MODE AND TAPE) <> 0 THEN
248   2                DO;
249   3                DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$TXE) = 0; END;
251   3                OUTPUT(SIO$STAT$PORT) = SIO$DTR$ON;          /* DTR ON */
252   3                CALL SIO$MS$DELAY(40);
253   3                OUTPUT(SIO$STAT$PORT) = SIO$DTR$OFF;
254   3                DO II = 1 TO MAX$DELAY;
255   4                   IF (INPUT(SIO$STAT$PORT) AND SIO$RXRDY) <> 0 THEN GOTO READY2;
257   4                END;
258   3                GOTO ERROR;
259   3       READY2:
                       DO WHILE (INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0; END;
261   3                CHAR = INPUT(SIO$DATA$PORT) AND 7FH;
262   3                END;
263   2             ELSE IF (MODE AND SERIAL) <> 0 THEN
264   2                DO;
265   3       LOOP:    DO II = 1 TO MAX$DELAY;
266   4                   IF SIO$CHAR$RDY THEN GOTO READY;
268   4                END;
269   3                GOTO ERROR;
270   3       READY:   CALL SIO$CHECK$CONTROL$CHAR;
```

```
271   3                IF CHAR = 11H THEN GOTO LOOP; /* GET ANOTHER IF CTL-Q */
273   3                END;
                    ELSE
274   2                DO;
275   3                DO II = 1 TO MAX$DELAY;
276   4                   IF SIO$CHAR$RDY THEN CALL SIO$CHECK$CONTROL$CHAR;
278   4                   IF (INPUT(PI$PORTC$STAT) AND PI$IBF) <> 0 THEN GOTO READY1;
280   4                END;
281   3                GOTO ERROR;
282   3          READY1:  CHAR = NOT INPUT(PI$PORTB);
283   3                END;
284   2             RETURN CHAR;
285   2           END SIO$READ$CHAR;

286   1          SIO$READ$BYTE:
                    /* THIS ROUTINE READS TWO HEX BYTES AND RETURNS THEIR BINARY
                       BYTE VALUE. */
                    PROCEDURE BYTE;
287   2                DECLARE T BYTE;
288   2                T = LOW(SIO$HEX(SIO$READ$CHAR));
289   2                T = SHL(T,4) + LOW(SIO$HEX(SIO$READ$CHAR));
290   2                CHECK$SUM = CHECK$SUM + T;
291   2                RETURN T;
292   2             END;

293   1          SIO$READ$WORD:
                    /* THIS ROUTINE READS FOUR HEX BYTES AND RETURNS THEIR BINARY
                       WORD VALUE. */
                    PROCEDURE WORD;
294   2                DECLARE T BYTE;
295   2                T = SIO$READ$BYTE;
296   2                RETURN SHL(DOUBLE(T),8) + DOUBLE(SIO$READ$BYTE);
297   2             END;

298   1          SIO$WRITE$HEX$FILE:
                    /* THIS ROUTINE IS CALLED BY THE WRITE AND TRANSFER COMMANDS TO
                       COMPLETE DECODING THE COMMAND LINE AND OUTPUT A HEX FILE.
                       IT OUTPUTS LEADING NULLS, START ADDRESS RECORD (8086 ONLY),
                       EXTENDED ADDRESS RECORDS (8086 ONLY), DATA RECORDS, EOF RECORD,
                       AND TRAILING NULLS. */
                    PROCEDURE;
299   2                DECLARE (LEN,INDEX) WORD, START$REC BYTE;
300   2                DECLARE (FIRST,LAST) STRUCTURE (OFF WORD, SEG WORD);
301   2                CALL SIO$GET$ADDR(@ARG1,CS);
302   2                FIRST.SEG = ARG1.SEG AND 0F000H;
303   2                FIRST.OFF = ARG1.OFF + SHL(ARG1.SEG,4);
304   2                IF CARRY THEN
305   2                   DO;
306   3                   FIRST.SEG = FIRST.SEG + 1000H;
307   3                   IF CARRY THEN GOTO ERROR;
309   3                   END;
310   2                IF CHAR<>',' THEN GOTO ERROR;
312   2                CALL SIO$GET$CHAR;
313   2                IF MODE$8086 THEN
314   2                   DO;
315   3                   CALL SIO$GET$ADDR(@ARG2,ARG1.SEG);
316   3                   LAST.SEG = ARG2.SEG AND 0F000H;
```

```
317   3                LAST.OFF = ARG2.OFF + SHL(ARG2.SEG,4);
318   3                IF CARRY THEN
319   3                   DO;
320   4                   LAST.SEG = LAST.SEG + 1000H;
321   4                   IF CARRY THEN GOTO ERROR;
323   4                   END;
                       /* CHECK IF END > START */
324   3                IF LAST.SEG < FIRST.SEG THEN GOTO ERROR;
326   3                IF LAST.SEG = FIRST.SEG THEN
327   3                   IF LAST.OFF < FIRST.OFF THEN GOTO ERROR;
                       /* CONVERT END ADDRESS FOR USE IN LOOPING */
329   3                LAST.SEG = ARG2.SEG + SHR(ARG2.OFF,4) - ARG1.SEG;
330   3                LAST.OFF = SHL(LAST.SEG,4) + (ARG2.OFF AND 0FH);
331   3                LAST.SEG = (LAST.SEG AND 0F000H) + ARG1.SEG;
332   3                END;
333   2             ELSE DO;
334   3                END$OFF = SIO$GET$WORD;
335   3                IF END$OFF<ARG1.OFF THEN GOTO ERROR;
337   3                LAST.SEG = ARG1.SEG;
338   3                LAST.OFF = END$OFF;
339   3                END;
340   2             IF CHAR <> ASCR THEN
341   2                DO;
342   3                START$REC = TRUE;
343   3                CALL SIO$GET$CHAR;
344   3                IF MODE$8086 THEN CALL SIO$GET$ADDR(@ARG3,CS);
346   3                ELSE ARG3.OFF = SIO$GET$WORD;
347   3                END;
                    ELSE
348   2                DO;
349   3                START$REC = FALSE;
350   3                ARG3.OFF = 0;
351   3                END;
352   2             IF CHAR<>ASCR THEN GOTO ERROR;
354   2             CALL SIO$CRLF;
355   2             CALL SIO$INIT$MODE;
356   2             CALL SIO$SECOND$DELAY;

357   2             DO I=1 TO 60;              /* LEADING NULLS */
358   3                CALL SIO$OUT$CHAR(0);
359   3             END;
360   2             CALL SIO$CRLF;

361   2             IF MODE$8086 THEN
362   2                DO;
363   3                IF START$REC THEN
364   3                   DO;
365   4                   CALL SIO$OUT$HEADER(04,0,03); /* START ADDRESS RECORD */
366   4                   CALL SIO$OUT$WORD(ARG3.SEG);
367   4                   CALL SIO$OUT$WORD(ARG3.OFF);
368   4                   CALL SIO$OUT$BYTE(CHECK$SUM);
369   4                   CALL SIO$CRLF;
370   4                   ARG3.OFF = 0;
371   4                   END;
372   3                END;

373   2       LOOP1:
```

```
                  IF MODE$8086 THEN DO;
375    3             CALL SIO$OUT$HEADER(02,0,02); /* EXTENDED ADDRESS RECORD */
376    3             CALL SIO$OUT$WORD(ARG1.SEG);
377    3             CALL SIO$OUT$BYTE(CHECK$SUM);
378    3             CALL SIO$CRLF;
379    3             END;
380    2             IF LAST.SEG = ARG1.SEG THEN END$OFF = LAST.OFF;
382    2             ELSE END$OFF = 0FFFFH;
383    2             LEN = STANDARD$LEN;              /* DATA RECORD */
384    2       LOOP: INDEX = END$OFF - ARG1.OFF;
385    2             IF INDEX<STANDARD$LEN-1 THEN LEN = INDEX+1;
387    2             CALL SIO$OUT$HEADER(LEN,ARG1.OFF,00);
388    2             DO I=1 TO LEN;
389    3                CALL SIO$OUT$BYTE$PTR(MEMORY$ARG1$PTR);
390    3                ARG1.OFF = ARG1.OFF + 1;
391    3             END;
392    2             CALL SIO$OUT$BYTE(CHECK$SUM);
393    2             CALL SIO$CRLF;
394    2             IF END$OFF <> ARG1.OFF-1 THEN GOTO LOOP;
396    2             IF LAST.SEG <> ARG1.SEG THEN
397    2                DO;
398    3                ARG1.SEG = ARG1.SEG + 1000H;
399    3                ARG1.OFF = 0;
400    3                GOTO LOOP1;
401    3                END;

402    2             CALL SIO$OUT$HEADER(00,ARG3.OFF,01);     /* EOF RECORD */
403    2             CALL SIO$OUT$BYTE(CHECK$SUM);
404    2             CALL SIO$CRLF;

405    2             DO I=1 TO 60;           /* TRAILING NULLS */
406    3                CALL SIO$OUT$CHAR(0);
407    3             END;
408    2             MODE = SERIAL;
409    2          END;


410    1       SIO$READ$HEX$FILE:
                  /* THIS ROUTINE IS CALLED BY THE READ AND LOAD COMMANDS TO
                     COMPLETE DECODING THE COMMAND LINE AND READ A HEX FILE. */
                  PROCEDURE;
411    2             DECLARE (REC$TYPE,LEN,I,T) BYTE, OFFSET WORD;
412    2             IF CHAR <> ASCR THEN CALL SIO$GET$ADDR(@ARG2,0);  /* GET BIAS ADDR */
414    2             ELSE ARG2.SEG,ARG2.OFF = 0;
415    2             ARG1.SEG = ARG2.SEG;  /* SEGMENT FOR 8080 FORMAT FILE */
416    2             IF CHAR<>ASCR THEN GOTO ERROR;
418    2             CALL SIO$CRLF;
419    2             CALL SIO$INIT$MODE;
420    2       LOOP:
                     DO WHILE SIO$READ$CHAR<>':';END;
422    2             CHECK$SUM = 0;
423    2             LEN = SIO$READ$BYTE;
424    2             OFFSET = SIO$READ$WORD;
425    2             ARG1.OFF = OFFSET + ARG2.OFF;
426    2             REC$TYPE = SIO$READ$BYTE;
427    2             IF REC$TYPE=03 THEN       /* START ADDR TYPE */
428    2                DO;
```

```
429   3              CS = SIO$READ$WORD;
430   3              IP = SIO$READ$WORD;
431   3              END;
432   2           IF REC$TYPE=02 THEN       /* EXTENDED ADDR TYPE */
433   2              ARG1.SEG = SIO$READ$WORD + ARG2.SEG;
434   2           IF REC$TYPE=01 THEN IF OFFSET <> 0 THEN IP = OFFSET;   /* EOF RECORD */
437   2           IF REC$TYPE=00 THEN       /* DATA TYPE */
438   2              DO I=1 TO LEN;
439   3                 T,MEMORY$ARG1 = SIO$READ$BYTE;
440   3                 IF MEMORY$ARG1<>T THEN GOTO ERROR;
442   3                 ARG1.OFF = ARG1.OFF + 1;
443   3              END;
444   2           T = SIO$READ$BYTE;        /* FETCH CHECKSUM */
445   2           IF CHECK$SUM<>0 THEN GOTO ERROR;
447   2           IF REC$TYPE<>01 AND LEN<>0 THEN GOTO LOOP;   /* EOF */
449   2           MODE = SERIAL;
450   2           CALL SIO$OUT$CHAR(0);  /* DELAY FOR LAST CR, LF SENT */
451   2           CALL SIO$OUT$CHAR(0);  /*      BY INTELLEC */
452   2        END;



            /*********************************************
            *   INTERRUPT AND RESTORE/EXECUTE SECTION    *
            *********************************************/

453   1     SAVE$REGISTERS:
            /* THIS ROUTINE IS USED TO SAVE THE STACKED USER'S REGISTERS IN THE
               MONITOR'S SAVE AREA. */
            PROCEDURE;
454   2           BP = MEMORY$USERSTACK;
455   2           USERSTACK.OFF = USERSTACK.OFF + 4;
456   2           DO I=0 TO 10;                    /* POP REGISTERS OFF OF STACK */
457   3              REG$SAV(REG$ORD(I)) = MEMORY$USERSTACK;
458   3              USERSTACK.OFF = USERSTACK.OFF + 2;
459   3           END;
460   2           SS = USERSTACK.SEG;
461   2           SP = USERSTACK.OFF;
462   2        END;

463   1     RESTORE$EXECUTE:
            /* THIS PROCEDURE RESTORES THE STATE OF THE USER MACHINE AND
               PASSES CONTROL BACK TO THE USER PROGRAM. IT CONTAINS A
               MACHINE LANGUAGE SUBROUTINE TO PERFORM THE POPPING OF THE
               USER REGISTERS AND TO EXECUTE AN 'IRET' TO TRANSFER CONTROL
               TO THE USER'S PROGRAM. */
            PROCEDURE;
464   2           DECLARE RESTORE$EXECUTE$CODE(*) BYTE DATA
                  (08BH,0ECH,           /* MOV BP,SP          */
                   08BH,046H,002H,      /* MOV AX,/BP/.PARM2  */
                   08BH,05EH,004H,      /* MOV BX,/BP/.PARM1  */
                   08EH,0D0H,           /* MOV SS,AX          */
                   08BH,0E3H,           /* MOV SP,BX          */
                   05DH,                /* POP BP             */
                   05FH,                /* POP DI             */
                   05EH,                /* POP SI             */
```

```
                        05BH,                   /* POP BX               */
                        05AH,                   /* POP DX               */
                        059H,                   /* POP CX               */
                        058H,                   /* POP AX               */
                        01FH,                   /* POP DS               */
                        007H,                   /* POP ES               */
                        0CFH),                  /* IRET                 */
                     RESTORE$EXECUTE$CODE$PTR WORD DATA (.RESTORE$EXECUTE$CODE);

465   2           USERSTACK.SEG = SS;
466   2           USERSTACK.OFF = SP;
467   2           DO I=0 TO 10;              /* PUSH USER'S REGISTERS ONTO HIS STACK */
468   3              USERSTACK.OFF = USERSTACK.OFF - 2;
469   3              MEMORY$USERSTACK = REG$SAV(REG$ORD(10-I));
470   3           END;
471   2           USERSTACK.OFF = USERSTACK.OFF - 2;
472   2           MEMORY$USERSTACK = BP;
473   2           CALL RESTORE$EXECUTE$CODE$PTR(USERSTACK.OFF,USERSTACK.SEG);
474   2        END;

475   1     INTERRUPT1$ENTRY:
           /* THIS PROCEDURE IS CALLED WHEN THE CPU IS INTERRUPTED BY EXECUTING
              AN INSTRUCTION WITH THE TRAP BIT SET (SINGLE STEP). */
           PROCEDURE INTERRUPT 1;
476   2           USERSTACK.OFF = STACKPTR;          /* CHANGE TO MONITOR'S STACK */
477   2           USERSTACK.SEG = STACKBASE;
478   2           STACKPTR =  MONITOR$STACKPTR;
479   2           STACKBASE = MONITOR$STACKBASE;
480   2           CALL SAVE$REGISTERS;
481   2           FL = FL AND (NOT STEP$TRAP);      /* CLEAR STEP FLAG */
482   2           IF LAST$COMMAND<>SS$COMMAND THEN  /* CONTINUE IF NOT SS */
483   2              CALL RESTORE$EXECUTE;
484   2           CSIP.OFF = IP;
485   2           CSIP.SEG = CS;
486   2           IF MEMORY$CSIP$PTR = INT3$PTR THEN
487   2              CALL RESTORE$EXECUTE;  /* EXIT TO PROCESS INT 3 */
488   2           CALL SIO$CRLF;
489   2           CALL SIO$UPDATE$IP;
490   2           IF CHAR=',' THEN
491   2              DO;
492   3              IP = CSIP.OFF;
493   3              CS = CSIP.SEG;
494   3              FL = FL OR STEP$TRAP;          /* SET STEP FLAG */
495   3              CALL RESTORE$EXECUTE;
496   3              END;
497   2           IF CHAR<>ASCR THEN  GOTO ERROR;
499   2           GOTO NEXT$COMMAND;
500   2        END;

501   1     INTERRUPT3$ENTRY:
           /* THIS PROCEDURE IS CALLED WHEN THE CPU EXECUTES A 'INT 3' INSTRUCTION.
              THE MONITOR INSERTS THIS (0CCH) FOR A BREAKPOINT.  ALSO A NMI
              INTERRUPT OR A USER SOFTWARE INTERRUPT MAY CAUSE THIS PROCEDURE TO BE
              CALLED. */
           PROCEDURE INTERRUPT 3;
502   2           USERSTACK.OFF = STACKPTR;
503   2           USERSTACK.SEG = STACKBASE;
```

```
504   2             STACKPTR =  MONITOR$STACKPTR;
505   2             STACKBASE = MONITOR$STACKBASE;
506   2             CALL SAVE$REGISTERS;
507   2             CALL SIO$CRLF;
508   2             GOTO AFTER$INTERRUPT;
509   2          END;

510   1       INTERRUPT32$ENTRY:
                  /* THIS ROUTINE IS EXECUTED WHEN THE CPU RECEIVES AN INTERRUPT FROM
                     THE 8259A. */    PROCEDURE INTERRUPT 32;
511   2             USERSTACK.OFF = STACKPTR;
512   2             USERSTACK.SEG = STACKBASE;
513   2             STACKPTR =  MONITOR$STACKPTR;
514   2             STACKBASE = MONITOR$STACKBASE;
515   2             CALL SAVE$REGISTERS;
516   2             CALL SIO$CRLF;
517   2             CALL SIO$OUT$CHAR('I');
518   2             CALL SIO$OUT$CHAR('=');
519   2             OUTPUT(IC$PORTA) = IC$OCW3;
520   2             CHAR = INPUT(IC$PORTA);
521   2             J = 1;
522   2             DO I = 0 TO 7;
523   3                IF (CHAR AND J) <> 0 THEN GOTO L1;
525   3                J = SHL(J,1);
526   3             END;
527   2       L1:   CALL SIO$OUT$BYTE(I);
528   2             CALL SIO$OUT$BLANK;
529   2             OUTPUT(IC$PORTA) = IC$EOI;
530   2             GOTO AFTER$INTERRUPT;
531   2          END INTERRUPT$32$ENTRY;

532   1       INIT$INT$VECTOR:
                  /* THIS ROUTINE INITIALIZES AN INTERRRUPT VECTOR AS FOLLOWS: THE OFFSET
                     FROM THE ADDRESS OF 'INT$ROUTINE' CORRECTED BY THE APPROPRIATE
                     NUMBER OF BYTES FOR THE INTERRUPT PLM PROLOGUE. THE SEGMENT FROM THE
                     CURRENT CS REGISTER IS DETERMINED BY A MACHINE LANGUAGE CODED
                     SUBROUTINE. */
                  PROCEDURE(INT$VECTOR$PTR,INT$ROUTINE$OFFSET);
533   2             DECLARE INT$VECTOR$PTR POINTER, INT$ROUTINE$OFFSET WORD,
                        VECTOR BASED INT$VECTOR$PTR STRUCTURE (OFF WORD, SEG WORD),
                        CORRECTION LITERALLY '19H',  /* OFFSET FOR PROLOGUE */
                        INIT$INT$VECTOR$CODE(*) BYTE DATA
                        (055H,                          /* PUSH BP            */
                         08BH,0ECH,                     /* MOV BP,SP          */
                         08CH,0C8H,                     /* MOV AX,CS          */
                         0C4H,05EH,004H,                /* LES BX,/BP/.PARM1  */
                         026H,089H,007H,                /* MOV ES:W/BX/,AX    */
                         05DH,                          /* POP BP             */
                         0C2H,004H,000H),               /* RET 4              */
                        INIT$INT$VECTOR$CODE$PTR WORD DATA (.INIT$INT$VECTOR$CODE);

534   2                CALL INIT$INT$VECTOR$CODE$PTR(@VECTOR.SEG);    /* SEGMENT PORTION */
535   2                VECTOR.OFF = INT$ROUTINE$OFFSET - CORRECTION;   /* OFFSET PORTION */
536   2          END;


            /* ************************************************************************
```

```
                  ************************************************************************

                                       COMMAND MODULE
                                       ==============
                  ABSTRACT
                  ========
                  THIS MODULE CONTAINS ALL THE COMMANDS IMPLEMENTED AS INDIVIDUAL PROCEDURES
                  AND CALLED FROM THE OUTER BLOCK OF THE COMMAND DISPATCH LOOP.

                  MODULE ORGANIZATION
                  ===================
                  THIS MODULE CONTAINS THE FOLLOWING SECTIONS:

                     1. COMMANDS SECTION
                         SIO$GO                GO
                         SIO$SINGLE$STEP       SINGLE STEP
                         SIO$EXAM$MEM          SUBSTITUTE MEMORY
                         SIO$EXAM$REG          EXAMINE REGISTER
                         SIO$MOVE              MOVE
                         SIO$DISPLAY           DISPLAY BYTES
                         SIO$COMPARE           COMPARE MEMORY
                         SIO$FIND              FIND BYTE/WORD
                         SIO$HEX$ARITH         PERFORM HEX ARITHMETIC
                         SIO$INPUT             INPUT PORT
                         SIO$OUTPUT            OUTPUT PORT
                         SIO$WRITE             WRITE DATA RECORDS
                         SIO$READ              READ DATA RECORDS
                         SIO$TRANSFER          TRANSFER FILE
                         SIO$LOAD              LOAD FILE
                       2. COMMAND DISPATCH (OUTER BLOCK, MAIN PROGRAM LOOP)
                         NEXT$COMMAND          DISPATCH
                         ERROR                 ERROR ROUTINE
               */

                  /************************
                  *   COMMAND SECTION    *
                  ************************/


537   1        SIO$GO:
                  /* IMPLEMENTS THE 'GO' COMMAND. THE USER MAY SPECIFY A NEW
                     IP:PC AND AN OPTIONAL BREAKPOINT. */
                  PROCEDURE;
538   2              CALL SIO$UPDATE$IP;
539   2              IF CHAR=',' THEN              /* BREAKPOINT */
540   2                 DO;
541   3                 CALL SIO$GET$CHAR;
542   3                 CALL SIO$GET$ADDR(@BRK1,CSIP.SEG);
543   3                 IF (CHAR<>ASCR) AND (CHAR<>',') THEN  GOTO ERROR;
545   3                 BRK1$SAVE = MEMORY$BRK1;
546   3                 MEMORY$BRK1 = BREAK$INST;
547   3                 IF MEMORY$BRK1<>BREAK$INST THEN GOTO ERROR;
549   3                 BRK1$FLAG = TRUE;
550   3                 IF CHAR = ',' THEN
551   3                    DO;
552   4                    CALL SIO$GET$CHAR;
553   4                    CALL SIO$GET$ADDR(@BRK2,CSIP.SEG);
```

```
554   4                  IF CHAR <> ASCR THEN GOTO ERROR;
556   4                  BRK2$SAVE = MEMORY$BRK2;
557   4                  MEMORY$BRK2 = BREAK$INST;
558   4                  IF MEMORY$BRK2 <> BREAK$INST THEN GOTO ERROR;
560   4                  BRK2$FLAG = TRUE;
561   4                  END;
562   3               END;
                      ELSE                          /* NO BREAKPOINT */
563   2                IF CHAR<>ASCR THEN  GOTO ERROR;
                      CALL SIO$CRLF;
566   2               IP = CSIP.OFF;
567   2               CS = CSIP.SEG;
568   2               FL = FL AND (NOT STEP$TRAP);       /* CLEAR IF SET */
569   2               CALL RESTORE$EXECUTE;
570   2            END;

571   1         SIO$SINGLE$STEP:
                  /* IMPLEMENTS THE SINGLE STEP COMMAND. DISPLAYS IP AND THE
                     CURRENT INSTRUCTION BTYE. OPENS CS:IP FOR INPUT. DEPRESSING
                     COMMA CAUSES THE MONITOR TO SINGLE STEP THE INSTRUCTION, AND
                     PERIOD TERMINATES THE COMMAND. */
                  PROCEDURE;
572   2               CALL SIO$UPDATE$IP;
573   2               IF CHAR<>',' THEN GOTO ERROR;
575   2               IP = CSIP.OFF;
576   2               CS = CSIP.SEG;
577   2               FL = FL OR STEP$TRAP;
578   2               CALL RESTORE$EXECUTE;
579   2            END;


580   1         SIO$EXAM$MEM:
                  /* IMPLEMENTS THE EXAMINE MEMORY COMMAND. */
                  PROCEDURE;
581   2               DECLARE W WORD;
582   2               CALL SIO$TEST$WORD$MODE;
583   2               CALL SIO$GET$ADDR(@ARG1,CS);
584   2               IF CHAR<>',' THEN  GOTO ERROR;
586   2               DO WHILE TRUE;
587   3                  CALL SIO$OUT$BLANK;
588   3                  IF WORD$MODE THEN
589   3                     CALL SIO$OUT$WORD(MEMORY$WORD$ARG1);
                        ELSE
590   3                     CALL SIO$OUT$BYTE$PTR(MEMORY$ARG1$PTR);
591   3                  CALL SIO$OUT$CHAR('-');
592   3                  CALL SIO$OUT$BLANK;
593   3                  CALL SIO$GET$CHAR;
594   3                  IF CHAR=ASCR THEN RETURN;
596   3                  IF CHAR<>',' THEN
597   3                     DO;
598   4                     W = SIO$GET$WORD;
599   4                     IF (CHAR <> ',') AND (CHAR <> ASCR) THEN GOTO ERROR;
601   4                     IF WORD$MODE THEN
602   4                        DO;
603   5                        MEMORY$WORD$ARG1 = W;
604   5                        IF MEMORY$WORD$ARG1<>W THEN GOTO ERROR;
```

```
606   5                    END;
                        ELSE
607   4                    DO;
608   5                    MEMORY$ARG1 = LOW(W);
609   5                    IF MEMORY$ARG1<>LOW(W) THEN GOTO ERROR;
611   5                    END;
612   4                 END;
613   3              IF CHAR=ASCR THEN RETURN;
615   3              IF WORD$MODE THEN
616   3                 ARG1.OFF = ARG1.OFF + 2;
                     ELSE
617   3                 ARG1.OFF = ARG1.OFF + 1;
618   3              CALL SIO$CRLF;
619   3              CALL SIO$OUT$WORD(ARG1.OFF);
620   3           END;
621   2        END;

622   1     SIO$EXAM$REG:
           /* IMPLEMENTS THE EXAMINE REGISTER COMMAND. SCANS FOR A VALID
              REGISTER NAME AND DISPLAYS THE VALUE OF THAT REGISTER WHICH IS
              OPTIONALLY OPENED FOR INPUT. COMMA INCREMENTS TO NEXT REGISTER
              UNLESS IT IS 'FL' WHICH TERMINATES AS DOES CR. */
           PROCEDURE;
623   2          DECLARE (T,I) BYTE;
624   2          DECLARE SAVE WORD;
625   2          CALL SIO$SCAN$BLANK;
626   2          IF CHAR=ASCR THEN
627   2             DO;
628   3             CALL SIO$CRLF;
629   3             DO I=0 TO 13;
630   4                CALL SIO$OUT$BLANK;
631   4                CALL SIO$OUT$CHAR(REG(I*2));
632   4                CALL SIO$OUT$CHAR(REG(I*2+1));
633   4                CALL SIO$OUT$CHAR('=');
634   4                CALL SIO$OUT$WORD(REG$SAV(I));
635   4                IF I=6 THEN CALL SIO$CRLF;
637   4             END;
638   3             RETURN;
639   3             END;
640   2          IF NOT(SIO$VALID$REG$FIRST) THEN  GOTO ERROR;
642   2          T = CHAR;
643   2          CALL SIO$GET$CHAR;
644   2          IF NOT(SIO$VALID$REG(T,CHAR)) THEN  GOTO ERROR;
646   2          I = REG$INDEX;
647   2          DO WHILE TRUE;
648   3             CALL SIO$OUT$CHAR('=');
649   3             CALL SIO$OUT$WORD(REG$SAV(I));
650   3             CALL SIO$OUT$CHAR('-');
651   3             CALL SIO$OUT$BLANK;
652   3             CALL SIO$GET$CHAR;
653   3             IF CHAR<>',' AND CHAR<>ASCR THEN
654   3                DO;
655   4                SAVE = SIO$GET$WORD;
656   4                IF (CHAR <> ',' ) AND (CHAR <> ASCR) THEN GOTO ERROR;
658   4                REG$SAV(I) = SAVE;
659   4                END;
660   3             IF CHAR=ASCR OR I=13 THEN RETURN;
```

```
662   3                I = I + 1;
663   3                CALL SIO$CRLF;
664   3                CALL SIO$OUT$CHAR(REG(I*2));
665   3                CALL SIO$OUT$CHAR(REG(I*2+1));
666   3             END;
667   2          END;

668   1       SIO$MOVE:
             /* IMPLEMENTS THE MOVE COMMAND. ACCEPTS 3 ARGUMENTS AMD MOVES THE
                BLOCK OF MEMORY SPECIFIED BY ARG1-ARG2 TO ARG3. ARG2<ARG1 OR THERE
                IS A DIFFERENCE WHEN THE BYTE IS READ BACK, THEN ERROR. */
             PROCEDURE;
669   2             CALL SIO$SCAN$BLANK;
670   2             CALL SIO$GET$ADDR(@ARG1,CS);           /* FIRST ARGUMENT */
671   2             IF CHAR<>',' THEN GOTO ERROR;
673   2             CALL SIO$GET$CHAR;
674   2             END$OFF = SIO$GET$WORD;                /* SECOND ARGUMENT */
675   2             IF END$OFF<ARG1.OFF THEN  GOTO ERROR;
677   2             IF CHAR<>',' THEN  GOTO ERROR;
679   2             CALL SIO$GET$CHAR;
680   2             CALL SIO$GET$ADDR(@ARG3,CS);     /* THIRD ARGUMENT */
681   2             IF CHAR<>ASCR THEN GOTO ERROR;
683   2             CALL SIO$CRLF;
684   2       LOOP:
                   MEMORY$ARG3 = MEMORY$ARG1;
685   2             IF MEMORY$ARG3<>MEMORY$ARG1 THEN GOTO ERROR;
687   2             IF ARG1.OFF = END$OFF THEN RETURN;
689   2             ARG1.OFF = ARG1.OFF + 1;
690   2             ARG3.OFF = ARG3.OFF + 1;
691   2             GOTO LOOP;
692   2          END;

693   1       SIO$DISPLAY:
             /* IMPLEMENTS THE DISPLAY BYTE COMMAND. IF CALLED WITH 1 PARM THEN
                 OUTPUTS A SINGLE BYTE. IF CALLED WITH 2 PARMS THEN OUTPUTS THE RANGE
                 BETWEEN THE TWO ADDRESSES. IF OFFSET<BEGIN THEN OUTPUTS ONLY A SINGLE
                 BYTE. */
             PROCEDURE;
694   2             DECLARE T BYTE;
695   2             CALL SIO$TEST$WORD$MODE;
696   2             CALL SIO$GET$ADDR(@ARG1,CS);
697   2             IF CHAR=ASCR THEN
698   2                END$OFF = ARG1.OFF;
                   ELSE
699   2                DO;
700   3                IF CHAR<>',' THEN GOTO ERROR;
702   3                CALL SIO$GET$CHAR;
703   3                END$OFF = SIO$GET$WORD;
704   3                IF END$OFF < ARG1.OFF THEN GOTO ERROR;
706   3                IF CHAR<>ASCR THEN  GOTO ERROR;
708   3                END;
709   2       NEWLINE:
                   CALL SIO$CRLF;
710   2             CALL SIO$OUT$WORD(ARG1.OFF);
711   2       LOOP: CALL SIO$OUT$BLANK;
712   2             IF WORD$MODE THEN
713   2                DO;
```

```
714   3                CALL SIO$OUT$WORD(MEMORY$WORD$ARG1);
715   3                IF ARG1.OFF = END$OFF THEN RETURN;
717   3                ARG1.OFF = ARG1.OFF + 1;
718   3                END;
                    ELSE
719   2                CALL SIO$OUT$BYTE$PTR(MEMORY$ARG1$PTR);
720   2                IF ARG1.OFF>=END$OFF THEN RETURN;
722   2                ARG1.OFF = ARG1.OFF + 1;
723   2                T = ARG1.OFF AND 000FH;
724   2                IF T=0 OR (WORD$MODE AND T=1) THEN GOTO NEWLINE;
726   2                GOTO LOOP;
727   2                END;

728   1          SIO$COMPARE:
                      /* IMPLEMENTS THE COMPARE COMMAND */
                      PROCEDURE;
729   2                CALL SIO$SCAN$BLANK;
730   2                CALL SIO$GET$ADDR(@ARG1,CS);
731   2                IF CHAR <> ',' THEN GOTO ERROR;
733   2                CALL SIO$GET$CHAR;
734   2                END$OFF = SIO$GET$WORD;
735   2                IF END$OFF < ARG1.OFF THEN GOTO ERROR;
737   2                IF CHAR <> ',' THEN GOTO ERROR;
739   2                CALL SIO$GET$CHAR;
740   2                CALL SIO$GET$ADDR(@ARG3,CS);
741   2                IF CHAR <> ASCR THEN GOTO ERROR;
743   2                CALL SIO$CRLF;
744   2          LOOP:
                      IF MEMORY$ARG1 <> MEMORY$ARG3 THEN
745   2                   DO;
746   3                   CALL SIO$OUT$WORD(ARG1.OFF);
747   3                   CALL SIO$OUT$BLANK;
748   3                   CALL SIO$OUT$BYTE$PTR(MEMORY$ARG1$PTR);
749   3                   CALL SIO$OUT$BLANK;
750   3                   CALL SIO$OUT$WORD(ARG3.OFF);
751   3                   CALL SIO$OUT$BLANK;
752   3                   CALL SIO$OUT$BYTE$PTR(MEMORY$ARG3$PTR);
753   3                   CALL SIO$CRLF;
754   3                   END;
755   2                IF ARG1.OFF = END$OFF THEN RETURN;
757   2                ARG1.OFF = ARG1.OFF + 1;
758   2                ARG3.OFF = ARG3.OFF + 1;
759   2                GOTO LOOP;
760   2             END SIO$COMPARE;

761   1          SIO$FIND:
                      /* IMPLEMENTS THE FIND COMMAND */
                      PROCEDURE;
762   2                DECLARE SEARCH$WORD WORD;
763   2                CALL SIO$TEST$WORD$MODE;
764   2                CALL SIO$GET$ADDR(@ARG1,CS);
765   2                IF CHAR <>  ',' THEN GOTO ERROR;
767   2                CALL SIO$GET$CHAR;
768   2                END$OFF = SIO$GET$WORD;
769   2                IF END$OFF < ARG1.OFF THEN GOTO ERROR;
771   2                CALL SIO$GET$CHAR;
772   2                SEARCH$WORD = SIO$GET$WORD;
```

```
773   2             IF CHAR <> ASCR THEN GOTO ERROR;
775   2             CALL SIO$CRLF;
776   2       LOOP:
                    IF WORD$MODE THEN
777   2                  DO;
778   3                  IF MEMORY$WORD$ARG1 = SEARCH$WORD THEN
779   3                     DO;
780   4                     CALL SIO$OUT$WORD(ARG1.OFF);
781   4                     CALL SIO$CRLF;
782   4                     END;
783   3                  END;
                       ELSE
784   2                  DO;
785   3                  IF MEMORY$ARG1 = LOW(SEARCH$WORD) THEN
786   3                     DO;
787   4                     CALL SIO$OUT$WORD(ARG1.OFF);
788   4                     CALL SIO$CRLF;
789   4                     END;
790   3                  END;
791   2                IF ARG1.OFF = END$OFF THEN RETURN;
793   2                ARG1.OFF = ARG1.OFF + 1;
794   2                GOTO LOOP;
795   2             END SIO$FIND;

796   1       SIO$HEX$ARITH:
                   /* IMPLEMENTS THE HEX ARITHMETIC COMMAND */
                   PROCEDURE;
797   2                DECLARE (W1,W2) WORD;
798   2                CALL SIO$SCAN$BLANK;
799   2                W1 = SIO$GET$WORD;
800   2                IF CHAR <> ',' THEN GOTO ERROR;
802   2                CALL SIO$GET$CHAR;
803   2                W2 = SIO$GET$WORD;
804   2                IF CHAR <> ASCR THEN GOTO ERROR;
806   2                CALL SIO$CRLF;
807   2                CALL SIO$OUT$WORD(W1+W2);
808   2                CALL SIO$OUT$BLANK;
809   2                CALL SIO$OUT$WORD(W1-W2);
810   2       END SIO$HEX$ARITH;

811   1       SIO$INPUT:
                   /* THIS ROUTINE IMPLEMENTS THE 'INPUT' COMMAND. USER SPECIFIES
                      A PORT AND THE DATUM OF THE PORT IS DISPLAYED. */
                 PROCEDURE;
812   2                DECLARE PORT WORD;
813   2                CALL SIO$TEST$WORD$MODE;
814   2                PORT = SIO$GET$WORD;
815   2       LOOP:
                       IF CHAR<>',' THEN GOTO ERROR;
817   2                CALL SIO$CRLF;
818   2                IF WORD$MODE THEN
819   2                   CALL SIO$OUT$WORD(INWORD(PORT));
                       ELSE
820   2                   CALL SIO$OUT$BYTE(INPUT(PORT));
821   2                CALL SIO$GET$CHAR;
822   2                IF CHAR=ASCR THEN RETURN;
824   2                GOTO LOOP;
```

```
825   2          END;

826   1       SIO$OUTPUT:
                 /* THIS ROUTINE IMPLEMENTS THE 'OUTPUT' COMMAND. THE USER SUPPLIED
                    DATUM IS OUTPUT TO THE SPECIFIED PORT. */
                 PROCEDURE;
827   2             DECLARE (DATUM,PORT) WORD;
828   2             CALL SIO$TEST$WORD$MODE;
829   2             PORT = SIO$GET$WORD;
830   2             IF CHAR<>',' THEN GOTO ERROR;
832   2             CALL SIO$GET$CHAR;
833   2       LOOP:
                    DATUM = SIO$GET$WORD;
834   2             IF CHAR=':' THEN GOTO ERROR;
836   2             IF WORD$MODE THEN
837   2                OUTWORD(PORT) = DATUM;
                    ELSE
838   2                OUTPUT(PORT) = LOW(DATUM);
839   2             IF CHAR=',' THEN
840   2                DO;
841   3                CALL SIO$CRLF;
842   3                CALL SIO$OUT$CHAR('-');
843   3                CALL SIO$OUT$BLANK;
844   3                CALL SIO$GET$CHAR;
845   3                IF CHAR <> ASCR THEN GOTO LOOP;
847   3                END;
848   2             RETURN;
849   2          END;

850   1       SIO$WRITE:
                 /* IMPLEMENTS THE PAPER TAPE WRITE COMMAND. */
                 PROCEDURE;
851   2             CALL SIO$GET$CHAR;
852   2             MODE$8086 = TRUE;
853   2             IF CHAR='X' THEN                  /* TEST FOR 8080 MODE */
854   2                DO;
855   3                MODE$8086 = FALSE;
856   3                CALL SIO$GET$CHAR;
857   3                END;
858   2             IF CHAR=ASBL THEN CALL SIO$GET$CHAR;
860   2             SAVE$MODE = TAPE OR SERIAL;
861   2             CALL SIO$WRITE$HEX$FILE;
862   2             RETURN;
863   2          END SIO$WRITE;

864   1       SIO$READ:
                 /* THIS PROCEDURE IMPLEMENTS THE PAPER TAPE READ COMMAND */
                 PROCEDURE;
865   2             CALL SIO$SCAN$BLANK;
866   2             SAVE$MODE = TAPE OR SERIAL;
867   2             CALL SIO$READ$HEX$FILE;
868   2             RETURN;
869   2          END SIO$READ;


870   1       SIO$TRANSFER:
                 /* THIS PROCEDURE IMPLEMENTS THE TRANSFER COMMAND */
```

```
                  PROCEDURE;
871    2             CALL SIO$GET$CHAR;
872    2             MODE$8086 = TRUE;
873    2             IF CHAR = 'X' THEN
874    2                DO;
875    3                MODE$8086 = FALSE;
876    3                CALL SIO$GET$CHAR;
877    3                END;
878    2             IF CHAR = ASBL THEN CALL SIO$GET$CHAR;
880    2             IF CHAR = 'S' THEN SAVE$MODE = SERIAL;
882    2             ELSE IF CHAR = 'P' THEN SAVE$MODE = PARALLEL;
884    2             ELSE GOTO ERROR;
885    2             CALL SIO$GET$CHAR;
886    2             IF CHAR <> ',' THEN GOTO ERROR;
888    2             CALL SIO$GET$CHAR;
889    2             CALL SIO$WRITE$HEX$FILE;
890    2             RETURN;
891    2          END SIO$TRANSFER;

892    1       SIO$LOAD:
                  /* THIS PROCEDURE IMPLEMENTS THE LOAD COMMAND */
                  PROCEDURE;
893    2             CALL SIO$SCAN$BLANK;
894    2             IF CHAR = 'S' THEN SAVE$MODE = SERIAL;
896    2             ELSE IF CHAR = 'P' THEN SAVE$MODE = PARALLEL;
898    2             ELSE GOTO ERROR;
899    2             CALL SIO$GET$CHAR;
900    2             IF CHAR = ',' THEN DO;
902    3                CALL SIO$GET$CHAR;
903    3                IF CHAR = ASCR THEN GOTO ERROR;
905    3                END;
906    2             ELSE IF CHAR <> ASCR THEN GOTO ERROR;
                     CALL SIO$READ$HEX$FILE;
909    2             RETURN;
910    2          END SIO$LOAD;

                  /****************************************
                  *   COMMAND DISPATCH MAIN PROGRAM LOOP   *
                  ****************************************/

911    1             DISABLE;
912    1             MODE = SERIAL;

                  /* THE FOLLOWING CODE DETERMINES THE BAUD RATE  OF THE SERIAL
                     INTERFACE BASED ON TWO 'U'S TYPED IN AT THE CONSOLE.  IT
                     INITIALIZES BOTH THE 8251A USART AND COUNTER 2 OF THE 8253
                     INTERVAL TIMER */

913    1             CALL SIO$RESET$USART;
914    1             OUTPUT(SIO$STAT$PORT) = SIO$CRT$MODE;
915    1             OUTPUT(IT$CONTROL$PORT) = IT$C2M3;
916    1             DO WHILE TRUE;
917    2                BRF = B9600;
918    2                OUTPUT(SIO$STAT$PORT) = SIO$CRT$CMD;
919    2                OUTPUT(IT$CTR2$PORT) = LOW(BRF);
920    2                OUTPUT(IT$CTR2$PORT) = HIGH(BRF);
921    2                DO II = 1 TO 1000;
```

```
 922   3                  CALL SIO$MS$DELAY(1);
 923   3                  IF SIO$CHAR$RDY THEN
 924   3                     DO;
 925   4                     IF (CHAR := INPUT(SIO$DATA$PORT)) = 80H THEN
 926   4                        DO;
 927   5                        BRF = B1200;
 928   5                        GOTO SBC$INIT5;
 929   5                        END;
 930   4                     CHAR = CHAR AND PARITY$MASK;
 931   4                     DO I = 0 TO 2;
 932   5                        IF CHAR = BR$CHAR(I) THEN GOTO SBC$INIT5;
 934   5                        ELSE BRF = 2*BRF;
 935   5                     END;
 936   4                     BRF = 2*BRF;
 937   4                     OUTPUT(IT$CTR2$PORT) = LOW(BRF);
 938   4                     OUTPUT(IT$CTR2$PORT) = HIGH(BRF);
 939   4                     CALL SIO$MS$DELAY(120);
 940   4                     CHAR = INPUT(SIO$DATA$PORT);
 941   4                     DO JJ = 1 TO 3000;
 942   5                        CALL SIO$MS$DELAY(1);
 943   5                        IF SIO$CHAR$RDY THEN
 944   5                           DO;
 945   6                           CHAR = INPUT(SIO$DATA$PORT) AND PARITY$MASK;
 946   6                           DO I = 0 TO 2;
 947   7                              IF CHAR = BR$CHAR(I) THEN GOTO SBC$INIT5;
 949   7                              ELSE BRF = 2*BRF;
 950   7                           END;
 951   6                           CALL SIO$RESET$USART;
 952   6                           OUTPUT(SIO$STAT$PORT) = SIO$TTY$MODE;
 953   6                           CHAR = 0;
 954   6                           OUTPUT(SIO$STAT$PORT) = SIO$TTY$CMD;
 955   6                           BRF = B110;
 956   6                           GOTO SBC$INIT5;
 957   6                           END;
 958   5                     END;
 959   4                     END;
 960   3               END;
 961   2            END;

 962   1      SBC$INIT5:
              OUTPUT(IT$CTR2$PORT) = LOW(BRF);
 963   1            OUTPUT(IT$CTR2$PORT) = HIGH(BRF);
 964   1            CALL SIO$MS$DELAY(200);
 965   1            CHAR = INPUT(SIO$DATA$PORT);
 966   1            CALL SIO$OUT$STRING(@SIO$SIGNON);

                  /* THE FOLLOWING CODE INITIALIZES THE 8259A.  THE STARTING ADDRESS
                     OF ITS 20H BYTE VECTOR TABLE IS 80H.  IT IS PROGRAMMED FOR
                     THE FULLY NESTED MODE.  ALL INTERRUPTS ARE SET UNMASKED. */

 967   1            OUTPUT(IC$PORTA) = IC$ICW1;
 968   1            OUTPUT(IC$PORTB) = IC$ICW2;
 969   1            OUTPUT(IC$PORTB) = IC$ICW4;
 970   1            OUTPUT(IC$PORTB) = IC$MASK;

                     /* INITIALIZE USER'S REGISTERS */
 971   1            CS,SS,DS,ES,FL,IP = 0;
```

```
 972   1            SP = USER$INIT$SP;

                    /* INITIALIZE INTERRUPT VECTORS */
 973   1            CALL INIT$INT$VECTOR(@INT$VECTOR(1),.INTERRUPT1$ENTRY);
 974   1            CALL INIT$INT$VECTOR(@INT$VECTOR(2),.INTERRUPT3$ENTRY);
 975   1            CALL INIT$INT$VECTOR(@INT$VECTOR(3),.INTERRUPT3$ENTRY);
 976   1            DO I = 32 TO 39;
 977   2               CALL INIT$INT$VECTOR(@INT$VECTOR(I),.INTERRUPT32$ENTRY);
 978   2            END;

 979   1            INT3$PTR = INT$VECTOR(3);  /* SAVE VECTOR 3 */
 980   1            BRK1$FLAG,BRK2$FLAG,SWITCH$BAUD = FALSE;

 981   1            MONITOR$STACKPTR = STACKPTR;
 982   1            MONITOR$STACKBASE = STACKBASE;

 983   1      NEXT$COMMAND:
                  /* THIS IS THE PERPETUAL COMMAND LOOP WHICH DISPATCHES TO EACH
                     COMMAND WHICH IS A SEPARATE PROCEDURE. */

                    CALL SIO$CRLF;
 984   1            CALL SIO$OUT$CHAR(0);
 985   1            CALL SIO$OUT$CHAR('.');
 986   1            IF SWITCH$BAUD THEN
 987   1               DO;
                     /* BAUD RATE WAS CHANGED FOR LOAD OR TRANSFER.  RESTORE
                        ORIGINAL BAUD RATE. */
 988   2               CALL SIO$SECOND$DELAY;
 989   2               SWITCH$BAUD = FALSE;
 990   2               CALL SIO$RESET$USART;
 991   2               OUTPUT(SIO$STAT$PORT) = SIO$CRT$MODE;
 992   2               OUTPUT(IT$CONTROL$PORT) = IT$C2M3;
 993   2               OUTPUT(SIO$STAT$PORT) = SIO$CRT$CMD;
 994   2               OUTPUT(IT$CTR2$PORT) = LOW(BRF);
 995   2               OUTPUT(IT$CTR2$PORT) = HIGH(BRF);
 996   2               END;

 997   1            CALL SIO$GET$CHAR;
 998   1            DO I=0 TO LAST(SIO$CMND);
 999   2               IF CHAR=SIO$CMND(I) THEN GOTO DISPATCH;
1001   2            END;
1002   1            GOTO ERROR;
1003   1       DISPATCH:
                    LAST$COMMAND = I;
1004   1            DO CASE I;
1005   2               CALL SIO$EXAM$MEM;
1006   2               CALL SIO$EXAM$REG;
1007   2               CALL SIO$GO;
1008   2               CALL SIO$SINGLE$STEP;
1009   2               CALL SIO$MOVE;
1010   2               CALL SIO$DISPLAY;
1011   2               CALL SIO$COMPARE;
1012   2               CALL SIO$FIND;
1013   2               CALL SIO$HEX$ARITH;
1014   2               CALL SIO$INPUT;
1015   2               CALL SIO$OUTPUT;
1016   2               CALL SIO$READ;
```

```
1017  2                CALL SIO$WRITE;
1018  2                CALL SIO$LOAD;
1019  2                CALL SIO$TRANSFER;
1020  2              END;
1021  1              GOTO NEXT$COMMAND;

1022  1        ERROR:
                 /* THIS ROUTINE HANDLES ALL ERRORS DETECTED BY THE MONITOR AND
                    WILL OUTPUT THE ERROR PROMPT TO THE OUTPUT PORT.*/

                 MODE = SERIAL;
1023  1              IF BRK1$FLAG THEN
1024  1                 DO;  /* ERROR IN ENTERING BREAKPOINT 2 */
1025  2                 MEMORY$BRK1 = BRK1$SAVE;
1026  2                 BRK1$FLAG = FALSE;
1027  2                 END;
1028  1              CALL SIO$OUT$CHAR('#');
1029  1              CALL SIO$MS$DELAY(200);
1030  1              GOTO NEXT$COMMAND;

1031  1        AFTER$INTERRUPT:
                 /* THIS ROUTINE IS CALLED AFTER AN INTERRUPT TO DISPLAY THE CS:IP
                    AND RESTORE BREAKPOINTED INSTRUCTION(S). */

                 IF BRK1$FLAG THEN
1032  1                 DO;
1033  2                 IF BRK2$FLAG THEN
1034  2                    DO;
1035  3                    MEMORY$BRK2 = BRK2$SAVE;
1036  3                    BRK2$FLAG = FALSE;
1037  3                    IF ((IP-1) AND 000FH)=(BRK2.OFF AND 000FH) AND
                              (SHR(IP-1,4)+CS)=(SHR(BRK2.OFF,4)+BRK2.SEG) THEN
1038  3                       DO;
1039  4                       IP=IP-1;
1040  4                       CALL SIO$OUT$STRING(@SIO$BREAK2$MSG);
1041  4                       END;
1042  3                    END;
1043  2                 MEMORY$BRK1 = BRK1$SAVE;
1044  2                 BRK1$FLAG = FALSE;
1045  2                 IF ((IP-1) AND 000FH)=(BRK1.OFF AND 000FH) AND
                              (SHR(IP-1,4)+CS)=(SHR(BRK1.OFF,4)+BRK1.SEG) THEN
1046  2                    DO;
1047  3                    IP = IP - 1;
1048  3                    CALL SIO$OUT$STRING(@SIO$BREAK1$MSG);
1049  3                    END;
1050  2                 END;
1051  1              CALL SIO$OUT$CHAR('@');
1052  1              CALL SIO$OUT$WORD(CS);
1053  1              CALL SIO$OUT$CHAR(':');
1054  1              CALL SIO$OUT$WORD(IP);
1055  1              CALL SIO$OUT$BLANK;
1056  1              CSIP.SEG = CS;
1057  1              CSIP.OFF = IP;
1058  1              CALL SIO$OUT$BYTE$PTR(MEMORY$CSIP$PTR);
1059  1              GOTO NEXT$COMMAND;

1060  1          END MONITOR;       /* END OF MODULE */
```

```
            EOF


MODULE INFORMATION:

     CODE AREA SIZE      = 1792H   6034D
     CONSTANT AREA SIZE  = 0000H      0D
     VARIABLE AREA SIZE  = 012DH    301D
     MAXIMUM STACK SIZE  = 0042H     66D
     1647 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

REQUEST FOR READER'S COMMENTS

The Microcomputer Division Technical Publications Department attempts to provide documents that meet the needs of all Intel product users. This form lets you participate directly in the documentation process.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. _____

NAME _____ DATE _____
TITLE _____
COMPANY NAME/DEPARTMENT _____
ADDRESS _____
CITY _____ STATE _____ ZIP CODE _____

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS . . .

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

First Class
Permit No. 1040
Santa Clara, CA

BUSINESS REPLY MAIL

No Postage Stamp Necessary if Mailed in U.S.A.

Postage will be paid by:

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Attention: Technical Publications

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 987-8080

Printed in U.S.A.