intel®

APPLICATION
NOTE

AP-43

November 1978

Using The iSBC™ 957 Execution Vehicle For Executing 8086 Program Code

Joe Barthmaier
OEM Microcomputer Systems Applications

Using The iSBC™ 957 Execution Vehicle For Executing 8086 Program Code

Contents

I. INTRODUCTION

The iSBC 957 Intellec—iSBC 86/12 Interface and Execution Package contains the hardware and software required to interface an iSBC 86/12 Single Board Computer with an Intellec Microcomputer Development System. The iSBC 957 package gives the 8086 user the capability to develop software on an Intellec System and then debug this software on an iSBC 86/12 board using a program download capability and an interactive system monitor. The 8086 user has all the capabilities of the Intellec system at his disposal and has the powerful iSBC 86/12 system monitor commands to use for debugging 8086 programs.

The iSBC 86/12 board is an Intel 8086 based processor board which, in addition to the processor, contains 32K bytes of dual port RAM, sockets for up to 16K bytes of ROM/EPROM, a serial I/O port, 24 parallel I/O lines, 2 programmable counters, 9 levels of vectored priority interrupts, and an interface to the MULTIBUS™ system bus. The iSBC 957 package consists of monitor EPROMs for the iSBC 86/12 board, Loader software for the Intellec system, four (4) cable assemblies, assorted line drivers and terminators, and signal adapters. The iSBC 957 package provides the capability of downloading and uploading program and data blocks between an iSBC 86/12 board and an Intellec system. In addition, monitor commands and displays may be input and viewed from the Intellec system console. The iSBC 957 package, when used with the iSBC 86/12 board and an Intellec Microcomputer Development System, provides the user with the capability to edit, compile or assemble, link, locate, download, and interactively debug programs for the 8086 processor. The iSBC 957 package and the iSBC 86/12 board form an excellent "execution vehicle" for users developing software for the 8086 processor regardless of whether the users are 8086 component users or iSBC 86/12 board users. Using the iSBC 957 package 8086 programs may be debugged at the full 5 MHz speed of the processor. The recommended hardware for the execution vehicle is an iSBC 660 system chassis with an 8 card slot backplane and power supply, an iSBC 032 32K byte RAM memory board, the iSBC 957 package, and the iSBC 86/12 board.

This application note will describe how the iSBC 957 package may be used to develop and debug 8086 programs. First a description of the iSBC 86/12 board will be presented. Readers familiar with the iSBC 86/12 board may want to skip this section. Next follows a detailed description of the iSBC 957 package and the iSBC 86/12 system monitor commands. A program example of a matrix multiplication routine will then be presented. This example will contain both assembly language and PL/M-86 procedures. The steps required to compile, assemble, link, locate and debug the program code will be explained in detail. A typical debugging session using the iSBC 86/12 system monitor will be presented.

II. THE iSBC™ 86/12 SINGLE BOARD COMPUTER

The iSBC 86/12 Single Board Computer, which is a member of Intel's complete line of iSBC 80/86 computer products, is a complete computer system on a single printed-circuit assembly. The iSBC 86/12 board includes a 16-bit central processing unit (CPU), 32K bytes of dynamic RAM, a serial communications interface, three programmable parallel I/O ports, programmable timers, priority interrupt control, MULTIBUS control logic, and bus expansion drivers for interface with other MULTIBUS-compatible expansion boards. Also included is dual port control logic to allow the iSBC 86/12 board to act as a slave RAM device to other MULTIBUS masters in the system. Provision is made for user installation of up to 16K bytes of read only memory. Figure 1 contains a block diagram of the iSBC 86/12 board and in Appendix A is a simplified logic diagram of the iSBC 86/12 board.

Central Processing Unit

The central processor for the iSBC 86/12 board is Intel's 8086, a powerful 16-bit H-MOS device. The 225 sq. mil chip contains 29,000 transistors and has a clock rate of 5MHz. The architecture includes four (4) 16-bit byte addressable data registers, two (2) 16-bit memory base pointer registers and two (2) 16-bit index registers, all accessed by a total of 24 operand addressing modes for complex data handling and very flexible memory addressing.

Instruction Set — The 8086 instruction repertoire includes variable length instruction format (including double operand instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulation functions. The instruction set of the 8086 is a functional superset of the 8080A/8085A family and with

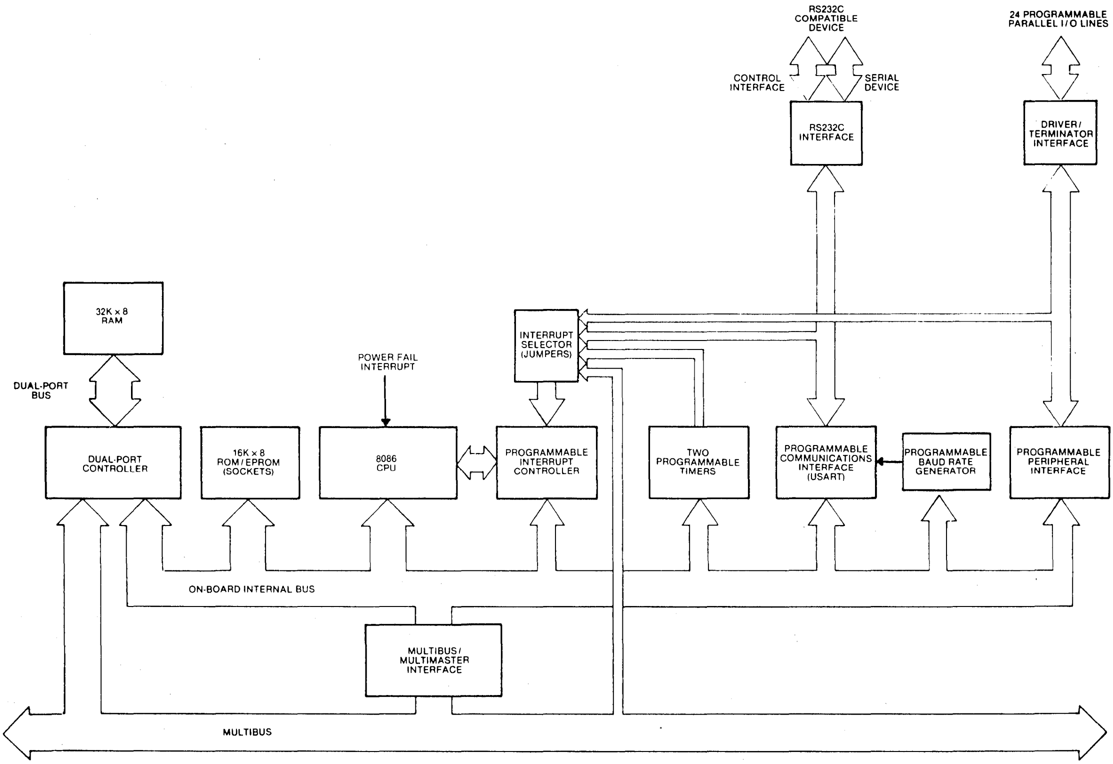

RS232C COMPATIBLE DEVICE
24 PROGRAMMABLE PARALLEL I/O LINES
CONTROL INTERFACE
SERIAL DEVICE
RS232C INTERFACE
DRIVER/ TERMINATOR INTERFACE
32K x 8 RAM
DUAL-PORT BUS
POWER FAIL INTERRUPT
INTERRUPT SELECTOR (JUMPERS)
DUAL-PORT CONTROLLER
16K x 8 ROM/EPROM (SOCKETS)
8086 CPU
PROGRAMMABLE INTERRUPT CONTROLLER
TWO PROGRAMMABLE TIMERS
PROGRAMMABLE COMMUNICATIONS INTERFACE (USART)
PROGRAMMABLE BAUD RATE GENERATOR
PROGRAMMABLE PERIPHERAL INTERFACE
ON-BOARD INTERNAL BUS
MULTIBUS/ MULTIMASTER INTERFACE
MULTIBUS

Figure 1. iSBC™ 86/12 Single Board Computer Block Diagram

available software tools, programs written for the 8080A/8085A can be easily converted and run on the 8086 processor.

Architectural Features — A 6-byte instruction queue provides pre-fetching of sequential instructions and can reduce the 1.2μsec minimum instruction cycle to 400 nsec by having the instruction already in the queue.

The stack oriented architecture facilitates nested sub-routines and co-routines, reentrant code and powerful interrupt handling. The memory expansion capabilities offer a 1 megabyte addressing range. The dynamic relocation scheme allows ease in segmentation of pure procedure and data for efficient memory utilization. Four segment registers (code, stack, data, extra) contain program loaded offset values which are used to map 16-bit addresses to 20-bit addresses. Each register maps 64K-bytes at a time and activation of a specific register is controlled explicitly by program control and is also selected implicitly by specific functions and instructions.

Bus Structure

The iSBC 86/12 board has an internal bus for communicating with on-board memory and I/O options, a system bus (the MULTIBUS) for referencing additional memory and I/O options, and the dual-port bus which allows access to RAM from the on-board CPU and the MULTIBUS System Bus. Local (on-board) accesses do not require MULTIBUS communication, making the system bus available for use by other MULTIBUS masters (i.e. DMA devices and other single board computers transferring to additional system memory). This feature allows true parallel processing in a multiprocessor environment. In addition, the MULTIBUS interface can be used for system expansion through the use of other 8- and 16-bit iSBC computers, memory and I/O expansion boards.

RAM Capabilities

The iSBC 86/12 board contains 32K-bytes of dynamic read/write memory. Power for the on-board RAM and refresh circuitry may be optionally provided on an auxiliary power bus, and memory protect logic is included for RAM battery backup requirements. The iSBC 86/12 board contains a dual port controller which allows access to the on-board RAM from the iSBC 86/12's CPU and from any other MULTIBUS master via the system bus. The dual port controller allows 8- and 16-bit accesses from the MULTIBUS System Bus and the on-board CPU transfers data to RAM over a 16-bit data path. Priorities have been established such that memory refresh is guaranteed by the on-board refresh logic and that the on-board CPU has priority over MULTIBUS requests for access to RAM. The dual-port controller includes independent addressing logic for RAM access from the on-board CPU and from the MULTIBUS system bus. The on-board CPU will always access RAM starting at location 00000_H. Address jumpers allow on-board RAM to be located starting on any 8K-byte boundary within a 1 megabyte address range for accesses from the MULTIBUS system bus. In conjunction with this feature, the iSBC 86/12 board has the ability to protect on-board memory from MULTIBUS access to any contiguous 8K-byte segments. These features allow multi-processor systems to establish local memory for each processor and shared system (MULTIBUS) memory configurations where the total system memory size (including local on-board memory) can exceed 1 megabyte without addressing conflicts.

EPROM/ROM Capabilities

Four sockets are provided for up to 16K-bytes of non-volatile read only memory on the iSBC 86/12 board. Configuration jumpers allow read only memory to be installed in 2K, 4K, or 8K increments.

On-board ROM is accessed via 16 bit data paths. System memory size is easily expanded by the addition of MULTIBUS compatible memory boards available in the iSBC 80/86 family.

Parallel I/O Interface

The iSBC 86/12 board contains 24 programmable parallel I/O lines implemented using the Intel 8255A Programmable Peripheral Interface. The system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports.

Therefore, the I/O interface may be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the flexibility of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 24 programmable I/O lines and signal ground lines are brought out to a 50-pin edge connector that mates with flat, woven, or round cable.

Serial I/O

A programmable communications interface using the Intel 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the iSBC 86/12 board. A software selectable baud rate generator provides the USART with all common communication frequencies. The USART can be programmed by the system software to select the desired asynchronous or synchronous serial data transmission technique (including IBM Bi-Sync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. The 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The RS232C compatible interface on each board, in conjunction with the USART, provides a direct interface to RS232C compatible terminals, cassettes, and asynchronous and synchronous modems. The RS232C command lines, serial data lines, and signal ground line are brought out to a 26 pin edge connector that mates with RS232C compatible flat or round cable. The iSBC 530 teletypewriter adapter provides an optically isolated interface for those systems requiring a 20 mA current loop. The iSBC 530 adapter may be used to interface the iSBC 86/12 board to teletypewriters or other 20 mA current loop equipment.

Programmable Timers

The iSBC 86/12 board provides three independent, fully programmable 16-bit interval timers/event counters utilizing the Intel 8253 Programmable Interval Timer. Each counter is capable of operating in either BCD or binary modes. Two of these timers/counters are available to the systems designer to generate accurate time intervals under software control. Routing for the outputs and gate/trigger inputs of two of these counters is jumper selectable. The outputs may be independently routed to the 8259A Programmable Interrupt Controller and to the I/O line drivers associated with the 8255A Programmable Peripheral Interface, or may be routed as inputs to the 8255A chip. The gate/trigger inputs may be routed to I/O terminators associated with the 8255A or as output connections from the 8255A. The third interval timer in the 8253 provides the programmable baud rate generator for the iSBC 86/12 RS232C USART serial port. In utilizing the iSBC 86/12, the systems designer simply configures, via software, each timer independently to meet system requirements. Whenever a given time delay or count is needed, software commands to the programmable timers/event counters select the desired function.

The contents of each counter may be read at any time during system operation with simple read operations for event counting applications, and special commands are included so that the contents can be ready "on the fly".

MULTIBUS™ and Multimaster Capabilities

The MULTIBUS system bus features asynchronous data transfers for the accommodation of devices with various transfer rates while maintaining maximum throughput. Twenty address lines and sixteen separate data lines eliminate the need for address/data multiplexing/demultiplexing logic used in other systems, and allow for data transfer rates up to 5 megawords/sec. A failsafe timer is included in the iSBC 86/12 board which can be used to generate an interrupt if an addressed device does not respond within 6 msec.

Multimaster Capabilities — The iSBC 86/12 board is a full computer on a single board with resources capable of supporting a great variety of OEM system requirements. For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically sharing system tasks through communication over the system bus), the iSBC 86/12 board provides full MULTIBUS arbitration control logic. This control logic allows up to three iSBC 86/12 boards or other bus masters, including iSBC 80 family MULTIBUS compatible 8-bit single board computers, to share the system bus in serial (daisy chain) priority fashion, and up to 16 masters to share the MULTIBUS with the addition of an external priority network. The MULTIBUS arbitration logic operates synchronously with a MULTIBUS clock (provided by the iSBC 86/12 board or optionally provided directly from the MULTIBUS System Bus) while data is transferred via a handshake between the master and slave modules. This allows different speed controllers to share resources on the same bus, and transfers via the bus proceed asynchronously. Thus, transfer speed is dependent on transmitting and receiving devices only. This design prevents slow master modules from being handicapped in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations, high speed direct memory access (DMA) operations, and high speed peripheral control, but are by no means limited to these three.

Interrupt Capability

The iSBC 86/12 board provides 9 vectored interrupt levels. The highest level is the NMI (Non-Maskable Interrupt) line which is directly tied to the 8086 CPU. This interrupt cannot be inhibited by software and is typically used for signalling catastrophic events (e.g., power failure).

The Intel 8259A Programmable Interrupt Controller (PIC) provides vectoring for the next eight interrupt levels.

The PIC accepts interrupt requests from the programmable parallel and serial I/O interfaces, the programmable timers, the system bus, or directly from peripheral equipment. The PIC then determines which of the incoming requests is of the highest priority, determines whether this request is of higher priority than the level currently being serviced, and, if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked, via software, by storing a single byte in the interrupt mask register of the PIC. The PIC generates a unique memory address for each interrupt level. These addresses contain unique instruction pointers and code segment offset values (for expanded memory operation) for each interrupt level. In systems requiring additional interrupt levels, slave 8259A PIC's may be interfaced via the MULTIBUS system bus, to generate additional vector addresses, yielding a total of 65 unique interrupt levels.

Interrupt Request Generation — Interrupt requests may originate from 16 sources. Two jumper selectable interrupt requests can be automatically generated by the programmable peripheral interface.

Two jumper selectable interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the CPU or a character is ready to be transmitted.

A jumper selectable request can also be generated by each of the programmable timers. Eight additional interrupt request lines are available to the user for direct interface to user designated peripheral devices via the system bus, and two interrupt request lines may be jumper routed directly from peripherals via the parallel I/O driver/terminator section.

Power-Fail Control

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the iSBC 635 Power Supply or equivalent.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards. High speed integer and floating point arithmetic capabilities may be added by using the iSBC 310 high speed mathematics unit. Memory may be expanded to 1 megabyte by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers. Modular expandable backplanes and cardcages are available to support multiboard systems.

III. THE iSBC™ 957 PACKAGE

The iSBC 957 Intellec—iSBC 86/12 Interface and Execution Package extends the software development capabilities of the Intellec Microcomputer Development systems to the Intel 8086 CPU. Programs for the 8086 may be written in PL/M-86 and/or assembly language and compiled or assembled on the Intellec system. These programs may then be downloaded from an Intellec ISIS-II disk file to the iSBC 86/12 board for execution and debug. The programs will execute at the full 5 MHz clock rate of the 8086 CPU with no speed degradation caused by the iSBC 957 hardware or software. Special communication software allows transparent access to the powerful interactive debug commands in the iSBC 86/12 monitor from the Intellec console terminal. These debug commands include single-step instruction execution, execution with breakpoints, memory and register displays, memory searches, comparison of two memory blocks and several other commands. After a debugging session, the debugged program code may be uploaded from the iSBC 86/12 board to an Intellec ISIS-II disk file.

The iSBC 957 Intellec—iSBC 86/12 Interface and Execution Package consists of the following:

a. Four Intel 2716 EPROMs which contain the system monitor program for the iSBC 86/12 board.

b. An ISIS-II diskette containing loader software for execution in the Intellec which provides for communications between the user or an Intellec ISIS-II file and the iSBC 86/12 board. Also included on the diskette are a library of routines for system console I/O.

c. Four cable assemblies used for transmitting commands, code and data between the iSBC 86/12 board and the Intellec system.

d. An iSBC 530 adapter assembly which converts serial communications signals from current loop to RS232C.

e. Line drivers and terminators used for the iSBC 86/12 parallel ports.

f. A small printed circuit board which is plugged into an iSBC 86/12 receiver/terminator socket and is used when program code is downloaded or uploaded using the parallel cable.

iSBC™—Intellec™ Configurations

There are two distinct functional configurations for the iSBC 957 package; one configuration for the Intellec Series II, Models 220 or 230 development systems and another for the Intellec 800 series development systems.

Intellec Series II System Configurations

When used with Intellec Series II Model 220 or 230 systems, a set of cables are used to connect the serial I/O port edge connector on the iSBC 86/12 board and the SERIAL 1 output port on the Intellec system. This configuration is shown in Figure 2. How this system functions is explained in the following paragraphs.

The SERIAL 1 port on the Intellec Series II Model 220 or 230 system is an RS232 port which is designed for use with a data terminal. This port may be used on the Intellec system for interfacing to RS232 devices such as CRT terminals or printers. The serial ports on the iSBC 86/12 board and the Intellec systems are connected as shown in the Figure 2. The flat ribbon cable connected to the iSBC 86/12 board has an edge connector for connecting to the board on one end and a standard RS232 connector on the other end. The second cable, the RS232 Up/Down Load cable, has an RS232 connector on each end. This cable, however,

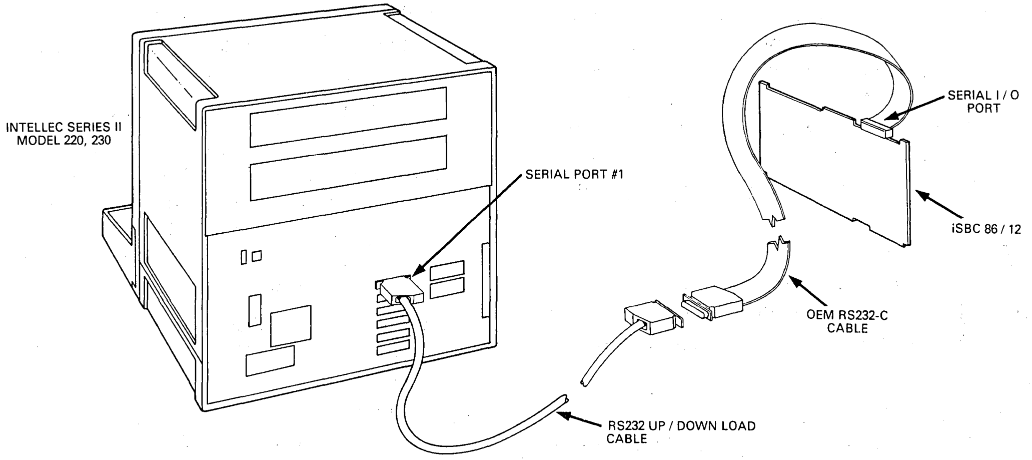


Figure 2. Intellec™ Series II Model 220, 230—iSBC™ 86 / 12 Configuration

is not a standard cable with the RS232 signals bussed between identically numbered pins on each of the connectors. The schematic for the cable is shown in Figure 3. Note that the TXD (transmit data) and the RXD (receive data) and the RTS (ready to send) and the CTS (clear to send) signals have been crossed. This is done because both the Intellec system and the iSBC 86/12 board are configured to act as data sets which are communicating with data terminals. Swapping these signals permits the units to communicate directly with no modifications to the Intellec or iSBC 86/12 systems themselves.

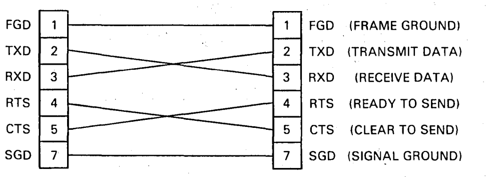


Figure 3. Intellec™—iSBC™ 86 / 12 RS232 UP / DOWN LOAD Cable

The software in the Intellec system accepts characters output from the iSBC 86/12 board through the Intellec SERIAL 1 port. The software then outputs these characters on the CRT monitor built into the Intellec Series II Model 220 or 230. In a similar fashion, characters input from the Intellec keyboard are passed down the serial link to the iSBC 86/12 monitor program. The integrated CRT monitor and keyboard on the Intellec system then becomes the "virtual terminal" of the iSBC 86/12 monitor program. If this were the only function of the iSBC 957 package, there would be no real benefit to the user. However, when the "virtual terminal" capability is combined with the capability to download and upload program code and data files between the Intellec ISIS-II file system and the iSBC 86/12 board, a very powerful software development tool is realized. The software in the Intellec system must examine the commands which are input from the keyboard and in the case of the LOAD and TRANSFER commands (see later sections for details on monitor commands), the software must open and read or write ISIS-II disk files.

Transfer rates using Intellec Series II Model 220 or 230 system are 9600 baud when transferring hexadecimal object files to or from a disk file and 600 baud when transferring commands between the iSBC 86/12 board and the CRT monitor and keyboard. With a 9600 baud transfer rate, it is possible to load 64K bytes of memory in about four minutes.

Intellec 800 System Configurations

The iSBC 957 package may be used with the Intellec 800 system in four different configurations. These four configurations are determined by two variables. The first variable is whether the iSBC 86/12 board is connected to the Intellec 800 TTY port or to the Intellec 800 CRT port. The second variable is whether or not a parallel cable is used for uploading and downloading hexadecimal object files. Figures 4A and 4B illustrate the four configurations.

In Figure 4A, the configuration shows the TTY port of the Intellec 800 system connected to the iSBC 86/12 serial port using two cables and an iSBC 530 teletypewriter adapter. The TTY port of the Intellec 800 system is designed for using a teletypewriter as the Intellec console device. To use this port for communication with the iSBC 86/12 board, the current loop TTY signal must be converted to an RS232 compatible voltage signal. This function is performed by the iSBC 530 adapter. The cable which connects the Intellec 800 system to the iSBC 530 adapter performs a function similar to the RS232 Up/Down Load cable described above. A schematic for this cable and all other components of the iSBC 957 package are included with the delivered product.

The transfer rate for both commands and data when the TTY port is connected to the iSBC 86/12 board is 110 baud. This means to download even moderately sized programs would require large amounts of time, several minutes or even hours. However, much faster times may be achieved by using the parallel ports of the iSBC 86/12 board and the Intellec system for downloading program files. This parallel port used on the Intellec 800 system is the output port labeled PROM which is normally used with the Universal Prom Pro-

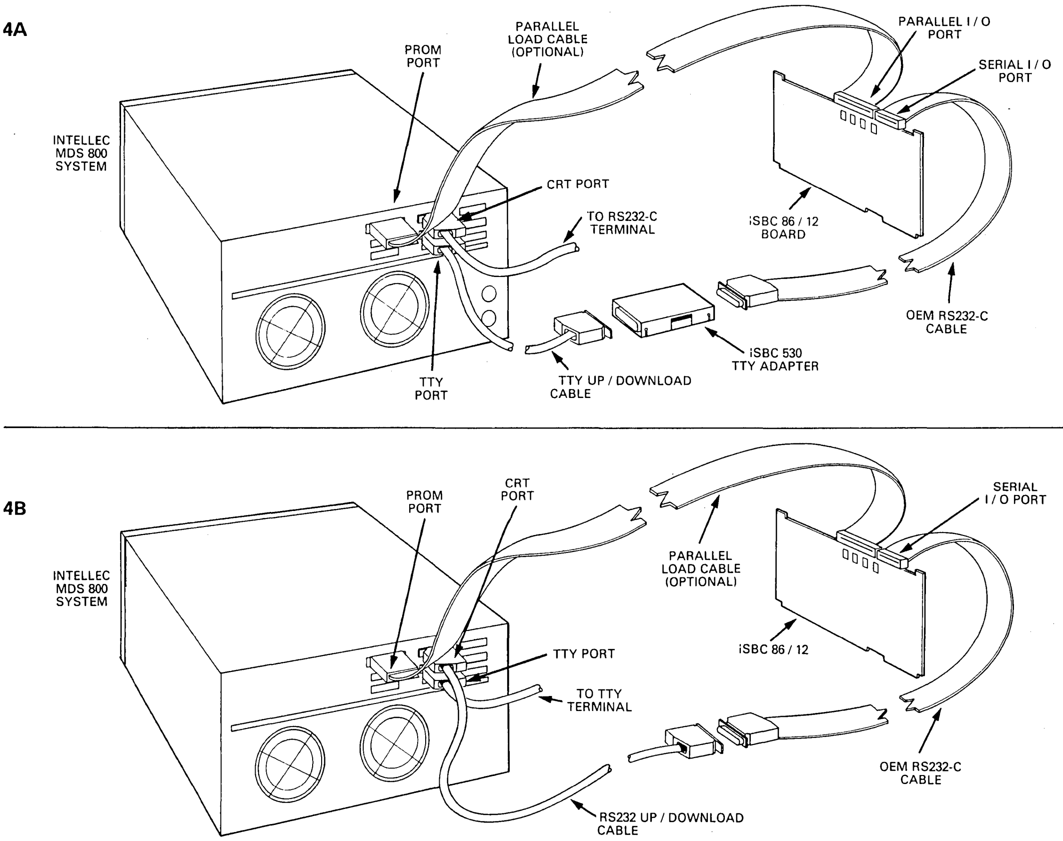


Figure 4A, 4B. Intellec™ 800—iSBC™ 86/12 Configurations

grammer. A cable is connected between the Intellec PROM port and the parallel I/O port, J1 of the iSBC 86/12 board. Parallel port B of the iSBC 86/12 board is used for the 8-bit byte transfers from the Intellec system to the iSBC 86/12 board, port A is used for the byte transfers from the iSBC 86/12 board to the Intellec system and port C is used for controlling the byte transfers. A special status adapter piggyback board must be inserted into a receiver/terminator socket of the iSBC 86/12 board. This status adapter circuit is required to provide the necessary handshaking signals from the iSBC 86/12 parallel ports to the Intellec PROM port.

The transfer rate achieved when downloading and uploading hexadecimal object files with the parallel cable is approximately 1,000 bytes per second. The time required to load 64K bytes of memory is approximately 2½ minutes.

Figure 4B shows a configuration with the Intellec 800 CRT port connected to the serial port of the iSBC 86/12 board. The TTY port of the Intellec 800 system is connected to a teletypewriter or some other current loop device to act as a system console. The optional parallel load cable is also shown. The cables used for this configuration are the same as those used with the Intellec Series II Configurations. Command transfer rates require 110 baud because the TTY port of the Intellec 800 system is used for communicating with the console device. However, hexadecimal object files can be loaded at 9600 baud since this operation uses only the Intellec to iSBC 86/12 RS232 link.

It is also possible to download files with the parallel cable, this mode being somewhat faster than the serial download mode (2½ minutes versus four minutes for 64K bytes of memory). Table I contains a summary of the command and memory transfer rates for each of the Intellec-iSBC 86/12 configurations.

Comparing the Intellec 800 configurations shown in Table 1 and in Figures 4A and 4B it should be noted:

1. Using the TTY port (Figure 4A) of the Intellec 800 system for communications with the iSBC 86/12 board (essentially) requires installation of the parallel cable and jumper modifications for downloading and uploading files, and thus, prevents the use of the parallel ports for other I/O functions.
2. Using the CRT port (Figure 4B) of the Intellec 800 system for communication with the iSBC 86/12 board provides for a fast serial download capability, thus freeing the parallel ports for other uses. However, this configuration requires a teletypewriter or a CRT capable of accepting a current loop input signal as the Intellec system console.

Table 1

COMMAND AND MEMORY TRANSFER RATES FOR INTELLEC—iSBC™ 86/12 CONFIGURATIONS

	INTELLEC SERIES II 220/230 SERIAL PORT TO iSBC 86/12	INTELLEC 800 TTY PORT TO iSBC 86/12	INTELLEC 800 CRT PORT TO iSBC 86/12
Effective Command Rate	600 Baud	110 Baud	110 Baud*
Load / Transfer Rate			
Serial	9600 Baud	110 Baud	9600 Baud
Parallel	N / A	1K bytes / sec**	1K bytes / sec**
Approximate Time to load 64K bytes of memory			
Serial	4 minutes	5 hours	4 minutes
Parallel	N / A	2.5 minutes	2.5 minutes

*The actual baud rate of the Intellec—iSBC 86/12 link is 9600 baud, but the effective command rate is determined by the slower Intellec— console serial link.

**Transmission rate over the parallel link is determined by the speed of the two processors and is approximately 1K bytes per second.

IV. THE iSBC 957—iSBC 86/12 MONITOR PROGRAM

The iSBC 86/12 monitor program is an EPROM resident program which facilitates debugging of user written programs. The monitor program used in the iSBC 86/12 board with the iSBC 957 package is the same monitor program written to interface the iSBC 86/12 directly to an RS232C data terminal. When interfaced directly to a terminal, the iSBC 86/12 board functions in a stand-alone environment communicating directly with the user via the data terminal. A user may use the monitor for entering small programs in hexadecimal format, executing a program, examining registers and memory contents, etc.

To use the monitor program with an Intellec system, the proper cables must be installed and the iSBC 957 Loader program must be loaded into Intellec memory and executed. The Loader program is resident on a file named SBC861, and when executed, the Loader outputs a sign-on message. Next, the iSBC 86/12 monitor program must be started and the baud rate of the iSBC 86/12 to Intellec serial communications link must be determined. This is done by pressing the RESET switch on the chassis

Table 2
MONITOR COMMAND LIST

COMMAND	FUNCTION AND SYNTAX
L Load Hex Object File	Loads hexadecimal object file from Intellec into iSBC 86/12 memory using serial (S) or parallel (P) mode. L{S|P},<*filename*>[,<*bias addr*>]<*cr*>
T Transfer Hex Object File	Transfers blocks of iSBC 86/12 memory to Intellec as a hex object file using serial (S) or parallel (P) mode. 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 with up to 2 optional breakpoints. G[<*start addr*>] [,<*break 1 addr*> [,<*break 2 addr*>]]<*cr*>
S Substitute Memory	Displays/modifies memory locations in byte or word format. 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 in byte or word format. 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 byte or word 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 byte or word data from input port. I[W]<*port addr*>,[,]*<*cr*>
O Port Output	Outputs byte or word data to output port. O[W]<*port addr*>,<*data*>[,<*data*>]*<*cr*>

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

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.

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.

containing the iSBC 86/12 board and typing two "U"s on the Intellec console. The ASCII uppercase character U has a binary pattern of alternating ones and zeros, the iSBC 86/12 monitor uses this pattern to determine the baud rate of the serial link. After the baud rate has been determined, the monitor program outputs a sign-on message to the console. An example of loader program execution and monitor program initialization is shown below (user entered characters are underlined).

```
:F1:SBC861
ISIS-II iSBC 86/12 LOADER, Vx.x
(user resets iSBC 86/12 board and types two "U"s)
iSBC 86/12 MONITOR, Vy.y
.
```

The monitor prompts with a period "." when it is ready for a command. The user can then enter a command file, 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. The user can cancel a command before entering the command terminator by pressing any illegal key (e.g., rubout or Control-X).

Table 2 contains a summary of the loader and monitor commands. These commands will not be explained in detail; instead, the next section of the application note will show examples of using these loader and monitor commands. The iSBC 957 User's Guide referenced at the front of this document does, however, contain a complete description of each of the monitor and loader commands.

Table 3 contains a list of the 8086 hardware registers and abbreviations used by the monitor program.

Table 3
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

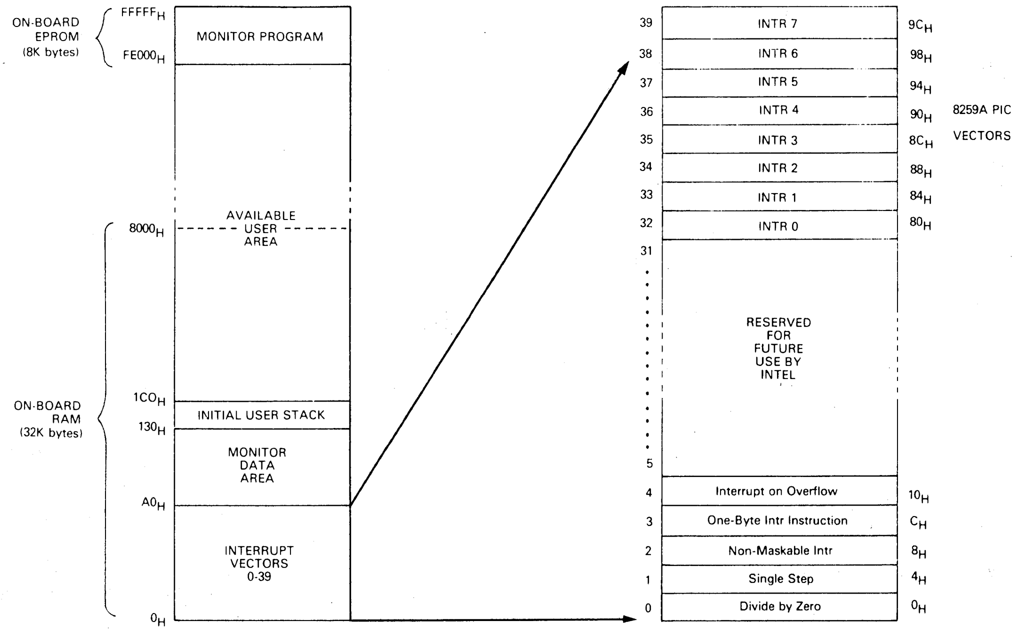


Figure 5. Memory Map of iSBC™ 86/12 Memory With Monitor Program

Figure 5 contains a memory map of the iSBC 86/12 memory with the monitor program. Note that the monitor uses the top 8K bytes of memory for its program code and the first 384 bytes of memory (locations Ø hex to 17F hex) for monitor and user stack, data and interrupt vectors. When the monitor program is reset, the segment registers, the IP and the flags are set to Ø; and the SP is set to ØlCØH allowing 64 bytes for the user's stack. If 64 bytes is not sufficient for the user's application program, the SP should be set to some other value. The monitor program sets the single-step, one-byte instruction trap and non-maskable interrupt vectors to monitor entry points. The monitor also sets the 8259A Priority Interrupt Controller to fully nested mode with level Ø at the highest priority and all interrupts unmasked. The eight interrupt vector addresses for the 8259A are also set to addresses in the monitor. User programs may change the 8259A interrupt vectors to interrupt service routine addresses within the user programs; it is not necessary for users to program the 8259A chip directly. When an interrupt occurs, control passes to either the monitor or directly to user code depending on the address stored in the vector location. When the monitor responds to an interrupt, it acknowledges the interrupt and displays the interrupt level, CS and IP register values and next instruction byte on the system console (e.g., I = 3 @ 100:230F F5).

When a user requests a breakpoint with a "G" command, the monitor inserts the single byte instruction trap instructions (INT 3) in the location where the breakpoint is requested. It is also possible for the user to code an INT 3 instruction in his program. When a user coded INT 3 instruction is executed, the monitor will be re-entered and a line with the format @<CS Value>:<IP Value> <Instruction byte> will be displayed (e.g., @1200:3FO2 F5).

Included on the diskette with the Loader program are two libraries containing I/O routines for the console. The library files are named SBCIOS.LIB and SBCIOL.LIB; they contain similar routines. 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 console input output routines, CI and CO, contained in the library should be used when performing character input and output on the console. Example PL/M-86 calls to the two routines are:

```
CI: PROCEDURE BYTE EXTERNAL;
    END CI;
CO: PROCEDURE (X) EXTERNAL;
    DECLARE X BYTE;
    END CO;
        .
        .
DECLARE INPUT$CHAR,
          OUTPUT$CHAR BYTE;
        .
        .
        .
INPUT$CHAR = CI;
        .
        .
        .
CALL CO(OUTPUT$CHAR);
        .
        .
        .
```

General Comments on Use of the iSBC 957 Package

1. If the iSBC 86/12 board is reset any time after the initial baud rate search, it is not necessary to reload the iSBC 957 Loader program or to download the program code a second time to the iSBC 86/12 board. It is only necessary to re-establish the communications link by typing two "U"s for the baud rate search.
2. The iSBC 86/12 board should not be plugged into an available card slot in an Intellec chassis; a separate chassis should be used. There are at least three reasons for this:
   a. There is only one RESET signal available on the Intellec system bus. Thus, each processor may not be reset independently. This means that the iSBC 86/12 board cannot be reset without re-booting the ISIS-II operating system and restarting the iSBC 957 Loader.
   b. The Intellec system uses five of the eight available interrupts on the system bus. This severely restricts the range of interrupts available to the iSBC 86/12 board. Also, the iSBC 86/12 board cannot turn-off the interrupt lamps on the Intellec front panel.
   c. The iSBC 86/12 board may address up to 1 Megabyte of memory using a 20 bit address. Many Intellec systems contain boards which generate and decode only the low order 16 address bits. For example, the iSBC 016 memory expansion board and the Intellec 800 monitor PROMs only decode 16 address bits. Memory expansion above 64K bytes in these systems is difficult since the boards which decode only 16 bits will force "holes" in the address space above 64K.
3. The iSBC 86/12 board is delivered with two inputs to the 8259A Priority Interrupt Controller connected. Interrupt request 2 (IR2) is connected to the counter Ø output of the 8253 Programmable Interval Timer. IR5 is connected to the INT5/signal of the MULTIBUS System Bus. If these interrupts are not desired, the wire wrap jumpers making the connections should be removed from the iSBC 86/12 board. A particular problem may exist with the counter Ø connection to IR2. If the 8253 counter Ø is not specifically initialized with software, a low frequency square wave output will exist at counter Ø's output. This may cause unwanted interrupts when interrupts are enabled by user programs.
4. If the iSBC 86/12 board is used in a system with expansion boards, it is important that the MULTIBUS bus exchange pins be properly jumpered. For example, if the iSBC 86/12 board is used with an iSBC 032 expansion memory board in a system, the BPRN/ MULTIBUS pin for the iSBC 86/12 board should be grounded.

   In addition, if any interrupts are used with the iSBC 86/12 board the BPRN/ pin must be grounded. This is true in both single and multiple board systems.
5. Certain user systems require more than one single board computer in the system for performing the functions required by the application. The MULTIBUS System Bus has been specifically designed to permit multiple CPU boards to communicate and to share system resources. However, debugging systems with multiple CPUs has always posed somewhat of a problem. The iSBC 957 package provides a solution to this problem. The serial cable which connects the iSBC 86/12 board to the Intellec system may be removed after the program has been downloaded to the iSBC 86/12 board. A console CRT may then be connected directly to the iSBC 86/12 board and the monitor program may be used to debug the program running on the board. Other iSBC 86/12 boards may also be downloaded from the Intellec system and then switched to their own local terminals. An 8-bit processor board, such as the iSBC 80/30 board, may also be included

in the system and ICE-85™ may be used for debugging the iSBC 80/30 program concurrently with the iSBC 86/12 programs. Using this scheme, it is possible to debug a system which has several CPU boards by setting breakpoints and using other debugging features on each of the individual CPUs.

V. MATRIX MULTIPLICATION EXAMPLE

To illustrate how the iSBC 957 package can be used to assist in the writing and debugging of 8086 programs on the iSBC 86/12 board, an example program of a matrix multiplication will be presented. The example chosen has been intentionally kept simple and straightforward. The emphasis of this section will be to document the steps required to assemble, compile, link, locate and debug software using an Intellec system, the iSBC 957 package and the iSBC 86/12 board. Part of the example will be written in 8086 assembly language and part in PL/M-86.

The main program is written in PL/M-86. The main program first performs some initialization and the matrix multiplication, then the program calls an assembly language procedure (subroutine), a PL/M-86 procedure and the console output procedure CO supplied in the I/O library on the iSBC 957 diskette. A flow diagram for the example program is shown in Figure 6.

Explanation of the Program Code

The program code is contained in three software modules EXECUTION$VEHICLE, FIND, and SBCCO. EXECUTION$VEHICLE contains the main program coded in PL/M-86 and the binary to ASCII conversion procedure BIN$DEC$ASC also coded in PL/M-86. The module FIND contains the assembly language procedure FIND$MX which searches a matrix for its maximum value. The module SBCCO resides in the library of console I/O routines supplied with the iSBC 957 package. The procedure CO will be used from this library.

The program code for the EXECUTION$VEHICLE and FIND modules will be explained in the following paragraphs. Appendix B contains compilation and assembly listings for the two modules; also contained in Appendix B is a memory and debug map for the linked modules. The listings contain circled reference letters (e.g., Ⓐ) which are referred to by the code description below. The listings in the appendix have been printed on fold-out pages so that they may easily be seen when reading the text.

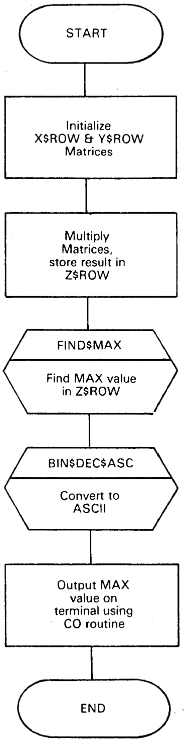

START
Initialize X$ROW & Y$ROW Matrices
Multiply Matrices, store result in Z$ROW
FIND$MAX
Find MAX value in Z$ROW
BIN$DEC$ASC
Convert to ASCII
Output MAX value on terminal using CO routine
END

Figure 6.
Flow Diagram of Matrix Multiplication Example

Much of the description given below assumes that the reader is familiar with the PL/M-86 language and compiler, the 8086 assembler, and the link and locate program QRL86. It is recommended that the reader have at least a cursory knowledge of these subjects. The Intel literature for these subjects is listed near the front of this application note.

The EXECUTION$VEHICLE Module

Ⓐ The first section of the module includes introductory comments and then statements to declare the matrices, other variables, and procedures used in the program. Note that the matrix dimensions are declared using the literals M, N, and P which are initially set to 6, 5, and 3. Later in this note, other values for M, N, and P will be used.

Ⓑ The next section of code contains the statements which initialize the two matrices that will be multiplied X$ROW and Y$ROW.

As a result of this initialization, the two matrices will contain values as shown in Figure 7.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & -2 \\ 0 & -1 & -2 \\ 0 & -1 & -2 \\ 0 & -1 & -2 \\ 0 & -1 & -2 \end{bmatrix}$$

X$ROW (6X5) Y$ROW (5X3)

Figure 7.
X$ROW and Y$ROW Matrices After Initialization

Ⓒ The next program section performs the matrix multiplication. The algorithm required to multiply two matrices X and Y, storing the result in a third matrix Z is:

$$Z_{mp} = \sum_{i=1}^{n} X_{mi} * Y_{ip}$$

Assuming X to be 6X5 matrix and Y a 5X3 matrix then

$$Z_{11} = X_{11}Y_{11} + X_{12}Y_{21} + X_{13}Y_{31} + X_{14}X_{41} + X_{15}Y_{51}$$

Thus, the upper left term is equal to the sum of the products of the top row of the X matrix times the left column of the Y matrix. The result that is obtained by multiplying the two matrices X$ROW and Y$ROW after they are initialized as explained above, is shown in Figure 8.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -5 & -10 \\ 0 & -10 & -20 \\ 0 & -15 & -30 \\ 0 & -20 & -40 \\ 0 & -25 & -50 \end{bmatrix}$$

Z$ROW (6X3)

Figure 8. Result of Multiplying the Initialized Matrices X$ROW and Y$ROW

Ⓓ The external assembly language procedure FIND$MX is called to determine the maximum value in the matrix. The procedure is a typed procedure and returns the maximum value to the calling program which stores it in the integer variable MAX.

Ⓔ The maximum value is then converted to a six (6) digit ASCII character string by the procedure BIN$DEC$ASC. The character string is stored in the array MAX$ASC$ARRAY, which contains the sign of the number and five (5) digits for the magnitude.

Ⓕ Finally, the characters "MAX VALUE =" are output on the system console followed by the 6 ASCII characters containing the maximum value. The PL/M-86 built-in procedure SIZE returns the number of bytes of the array TEXT as a word value. The PL/M-86 built-in procedure SIGNED changes the type of the value from WORD to INTEGER. This is required so that the type of the arguments in the DO statement agree. The console output procedure CO is used to output the characters on the system console.

Ⓖ Also contained in the module MATRIX.PLM is the binary to ASCII conversion procedure BIN$DEC$ASC. The first portion of the code contains the comments explaining the parameters and the calling sequence followed by the declarations. Note that the address of the array where the characters are to be stored is passed to the procedure and that the characters will be stored in the array using based variables. The next section of the code stores either a + or − sign in the first character position of the ASCII array and stores the absolute value of VALUE in the variable TEMP. Finally, the binary value is converted to ASCII using the algorithm explained in the comments. The MOD operator returns the remainder of the division by 10. The UNSIGN built-in procedure is required to change the type of the expression from INTEGER to WORD.

The FIND Module

Ⓗ The FIND module contains the assembly language procedure FINDMX. The calling sequence and the parameters are explained in the comments at the beginning of the listing. Note that the label FINDMX has been declared PUBLIC so the link program can fill in its address in the CALL statement in the main program of module EXECUTION$VEHICLE.

Ⓘ The FIND module will contain three segments: a data segment, a stack segment and a code segment. It will be both convenient and pragmatic to append these three segments to the code, data and stack segments created by the compiler for the EXECUTION$VEHICLE module. To accomplish this, the three segments must be given the same SEGMENT and CLASS names as those given these segments by the compiler. The SEGMENT and CLASS names used by the compiler are CODE, DATA, and STACK. The GROUP statements are used to place the segments DATA and STACK in the group DGROUP and the segment CODE in the group CGROUP. These group definitions conform with the group definitions generated by the PL/M-86 compiler when the SMALL size control option is used. A group is a collection of segments which requires less than 64K bytes of memory.

The ASSUME directive informs the assembler that the DS and SS registers will contain the base address of DGROUP and the CS register will contain the base address of CGROUP. This information will be used by the assembler when constructing machine instructions.

Ⓙ The first segment appearing in the module is the data segment. The order of the segments is arbitrary, although it is recommended that the data segment precede the code segment to minimize forward references to variables which may cause the assembler to generate longer instruction codes. The data segment is declared PUBLIC, aligned on a WORD boundary and given both a segment and class name of DATA. Then follows the contents of the segment. In this particular example, only one word of storage is required. The ENDS directive indicates the end of the segment.

Ⓚ Next comes the stack segment which is given the segment name of STACK, the combine-type attribute of STACK and the class name of STACK. The combine-type attribute of STACK assures that the stack storage required in this module will be appended to the storage required in the PL/M-86 compiled modules. Two bytes of stack are required by the code in this module, however, the monitor uses 13 words of stack when breakpoints and interrupts are used. Therefore, 14 words are reserved for the stack.

Ⓛ Finally comes the code segment. The code segment has been given a segment name and class name of CODE and a group name of CGROUP, and has been declared PUBLIC. The alignment attribute of BYTE is specified since it is desired that the code from this module be appended directly to the code from other modules without gaps between the code modules.

The assembly language code follows next. The code for the procedure must be enclosed between a pair of PROC, ENDP statements. The PROC statement is given the label FINDMX and specified as a NEAR procedure indicating it will be called with a near (intra-segment) CALL instruction and not a far (inter-segment) CALL instruction.

The comments at the beginning of the module and adjacent to the program statements explain the function being performed by the assembly language code.

The SBCCO Module

Ⓜ The console output procedure CO is contained in the object module SBCCO of the library file SBCIOS.LIB. SBCIOS.LIB is part of the iSBC 957 package I/O libraries. The calling sequence and parameters for CO may be seen in the external procedure declaration in the EXECUTION$VEHICLE module.

Compiling the EXECUTION$VEHICLE Module

The EXECUTION$VEHICLE module is stored on a file named MATRIX.PLM on disk device :F1:. To compile the module, the following command line is used:

```
– PLM86 :F1:MATRIX.PLM DEBUG
```

This command line will cause the module stored in the file :F1:MATRIX.PLM to be compiled. The object code generated will be stored in a file with the default name :F1:MATRIX.OBJ and the listing generated will be stored in a file with the default name :F1:MATRIX.LST. To override the default object and listing files, the NOOBJECT and NOLIST compiler control switches can be used. File names for the listing and object files may also be specified in the command line. The DEBUG compiler control switch causes the compiler to generate extra symbol and line number information which will be used during debugging of the program. A listing of the compiled EXECUTION$VEHICLE module is contained in Appendix B.

To aid in the debugging of the program, the module was compiled a second time with the following command line:

– PLM86 :F1:MATRIX.PLM NOOBJECT
CODE DEBUG PRINT (:F1:MATRIX.XLS)

This command line specified that no object file is to be created and a listing file should be stored in the file :F1:MATRIX.XLS. The CODE compiler control switch causes the compiler to list the assembly language statements which the compiler has generated for each line of PL/M code. The listing stored in the file MATRIX.XLS is contained in Appendix C.

Assembly of the FIND Module

The assembly language module FIND is stored on a file named FIND.ASM, to assemble this module the following command line is used:

ASM86 :F1:FIND.ASM DEBUG

This command line will cause the FIND module to be assembled with the object code stored in the default file :F1:FIND.OBJ and the listing stored in the default file :F1:FIND.LST. The listing of the assembled FIND module is contained in Appendix B.

Linking and Locating the Object Module

To link and locate the object modules, the QRL86 program will be used. The QRL86 program performs both the linking and the locating of the object modules in a single step. QRL86 is primarily designed for the debugging stages of program development. Some applications may require the extended capabilities of the separate LINK and LOCATE programs when the final link and locate is performed. The command line used to invoke the QRL86 program is:

QRL86 :F1:MATRIX.OBJ, :F1:FIND.OBJ,
SBCIOS.LIB ORIGIN (1000H)

This command line will cause QRL86 to link the code from the three modules and to locate the resultant absolute object module starting at location 1000 hexadecimal. The iSBC 86/12 monitor uses the first 180H bytes of memory for the monitor stack, data and interrupt vectors, 1000H was chosen as a convenient starting address for the program. The absolute object code will be stored in a default file :F1:MATRIX (note no file name extension is used). By default, the memory and debug maps which are generated are stored in the file :F1:MATRIX.MPQ and are contained in Appendix B.

Ⓝ The memory map contains the starting addresses and sizes of the CODE, CONST, DATA, STACK and MEMORY segments of the object module. Note that the start address for the program is specified as (Ø1ØØH, ØØØ2H) indicating a CS value of Ø1ØØH and an IP value of ØØØ2H or an absolute value of Ø1ØØ2H. The first two bytes of the code segment contain address values which the code generated by the compiler will use for setting up the DS and SS registers. The memory map shows the code segments from the three modules collected into the group CGROUP. The code segment from the EXECUTION$VEHICLE module is given the segment and class names of CODE and is put into CGROUP by the PL/M compiler. To assure that the code segment from the FIND module is concatenated with the code segment from the EXECUTION$VEHICLE module the identical class, segment and group names were specified in the SEGMENT and GROUP statements in the FIND module. Next, the group DGROUP is shown in the memory map. DGROUP contains 4 segments labelled CONST, DATA, STACK and MEMORY. Putting all of these segments in the same group tells the linker that they will all be in the same 64K block of memory. The SMALL size control option of the compiler, which was invoked by default, creates CGROUP, DGROUP, and the segments contained in them.

Ⓟ The debug map contains the memory address of variables, instruction labels and the addresses of each code line of the PL/M-86 module. Notice that the variable storage labels have their addresses specified in the format (DS register value, displacement). For example, the variable TEMP has an address of DS=Ø12AH, displacement = ØØØCH or an absolute address of Ø136H. Instruction labels and line numbers use the format (CS register value, IP register value). Thus, line number six (6) in the module EXECUTION$VEHICLE has the address CS=Ø1ØØH, IP=ØB5H or Ø11B5H.

Object to Hex Conversion

Before downloading the program to the iSBC 86/12, the format of the object module must be converted from the absolute object module format which QRL86 creates to a hexadecimal/ASCII representation of the object module. This is done using the program OH86 with the following command line:

OH86 :F1:MATRIX TO :F1:MATRIX.HEX

Downloading and Debugging the Program

The hardware configuration used for debugging the matrix multiplication example program code was an Intellec Series II Model 230 development system, the iSBC 957 package, an iSBC 86/12 board, and an iSBC 660 system chassis. What follows is the system-user dialog for a typical debugging session.

The first step required is to bootstrap load the ISIS-II operating system by hitting the RESET switch of the Intellec. The Intellec resident loader software is then loaded and executed. Throughout the dialog which follows operator entered characters will be underlined:

```
ISIS-II, V3.4
-SBC861

ISIS-II ISBC 86/12 LOADER, V1.2
.
```

To initialize the iSBC 86/12 monitor, the user must hit the RESET switch on the iSBC 660 chassis and type two "U"s on the system console. The monitor program will output a line on the console when it is properly initialized.

```
ISBC 86/12 MONITOR, V1.2
```

The monitor command "X" is typed to check that the monitor is properly operating and to examine the contents of the 8086 registers.

```
.X
 AX=ØØØØ BX=ØØØØ CX=ØØØØ DX=ØØØØ SP=Ø1CØ BP=ØØØØ SI=ØØØØ
 DI=ØØØØ CS=ØØØØ DS=ØØØØ SS=ØØØØ ES=ØØØØ IP=ØØØØ FL=ØØØØ
.
```

To download the hex object file to the iSBC 86/12, the "L" command is used. Because an Intellec Series II Model 230 is being used, a serial download is specified. The hex file name is MATRIX.HEX which is resident on disk device :F1:.

```
.LS,:F1:MATRIX.HEX
```

The "X" command is used again to examine the CPU registers. Note that the monitor has changed the contents of the CS and IP registers to the value of the starting address of the program.

```
 .X
 AX=ØØØØ BX=ØØØØ CX=ØØØØ DX=ØØØØ SP=Ø1CØ BP=ØØØØ SI=ØØØØ
 DI=ØØØØ CS=Ø1ØØ DS=ØØØØ SS=ØØØØ ES=ØØØØ IP=ØØØ2 FL=ØØØØ
.
```

The "D" command is next used to display the first 101 bytes of the program code. Unless another segment register is specified, the display command assumes all addresses specified are relative to the CS register. Thus, the code displayed will be from absolute addresses 1000 through 1100. The program code displayed may be compared with program code generated by the PL/M-86 compiler shown in Appendix C, code line 36.

```
 .DØ,1ØØ
ØØØØ 2A Ø1 FA 2E 8E 16 ØØ ØØ BC DØ ØØ 8B EC 16 1F FB
ØØ1Ø C7 Ø6 8E ØØ ØØ ØØ 81 3E 8E ØØ Ø5 ØØ 7E Ø3 E9 3C
ØØ2Ø ØØ C7 Ø6 9Ø ØØ ØØ ØØ 81 3E 9Ø ØØ Ø4 ØØ 7E Ø3 E9
ØØ3Ø 22 ØØ 8B Ø6 8E ØØ B9 ØA ØØ F7 E9 8B 36 9Ø ØØ D1
ØØ4Ø E6 89 C3 8B ØE 8E ØØ 89 88 1Ø ØØ 81 Ø6 9Ø ØØ Ø1
ØØ5Ø ØØ E9 D3 FF 81 Ø6 8E ØØ Ø1 ØØ E9 B9 FF C7 Ø6 8E
ØØ6Ø ØØ ØØ ØØ 81 3E 8E ØØ Ø4 ØØ 7E Ø3 E9 4Ø ØØ C7 Ø6
ØØ7Ø 9Ø ØØ ØØ ØØ 81 3E 9Ø ØØ Ø2 ØØ 7E Ø3 E9 26 ØØ 8B
ØØ8Ø Ø6 9Ø ØØ F7 D8 5Ø 8B Ø6 8E ØØ B9 Ø6 ØØ F7 E9 8B
ØØ9Ø 36 9Ø ØØ D1 E6 89 C3 59 89 88 4C ØØ 81 Ø6 9Ø ØØ
ØØAØ Ø1 ØØ E9 CF FF 81 Ø6 8E ØØ Ø1 ØØ E9 B5 FF C7 Ø6
ØØBØ 92 ØØ ØØ ØØ 81 3E 92 ØØ Ø2 ØØ 7E Ø3 E9 8C ØØ C7
ØØCØ Ø6 8E ØØ ØØ ØØ 81 3E 8E ØØ Ø5 ØØ 7E Ø3 E9 72 ØØ
ØØDØ 8B Ø6 8E ØØ B9 Ø6 ØØ F7 E9 8B 36 92 ØØ D1 E6 89
ØØEØ C3 C7 8Ø 6A ØØ ØØ ØØ C7 Ø6 9Ø ØØ ØØ ØØ 81 3E 9Ø
ØØFØ ØØ Ø4 ØØ 7E Ø3 E9 41 ØØ 8B Ø6 8E ØØ B9 ØA ØØ F7
Ø1ØØ E9
.
```

The PL/M-86 compiler ends the main program in the EXECUTION$VEHICLE module with a halt instruction. After execution of the program it is more desirable to return to the monitor. To accomplish this, an INT 3 instruction (code=CC) will be substituted for the halt instruction (code= F4) at the address of 1B4H relative to a CS value of 100H. First the "D" command is used to verify the address of the halt instruction, then the "S" command is used to change the instruction to an INT 3 instruction.

```
 .D1B4
Ø1B4 F4
 .S1B4, F4- CC
 .
```

To execute the PL/M-86 main program, the "G" command is used. After the "G" is typed, the current contents of the IP are output, followed by the contents of the byte pointed to by the IP. A new value for the IP or breakpoint addresses may be specified before a carriage return <CR> is typed. In this example, only a <CR> is typed.

```
 .G ØØØ2- FA
MAX VALUE = -ØØØ5Ø
@Ø1ØØ:Ø1B5 55
.
```

The program executes and outputs the maximum value of the matrix calculated. The INT 3 instruction is executed which causes a return to the monitor. The monitor types out an at-sign (@) followed by the CS and IP register values and the first byte of the instruction following the INT 3 instruction.

The "X" command is typed to examine the CPU registers. Note that the program has set both the SS and DS registers to Ø12A. (Ø12AØH is the address of the DGROUP as shown in the memory map.)

```
.X
 AX=ØØ3Ø BX=ØØØ5 CX=ØØØA DX=ØØØØ SP=ØØDØ BP=ØØDØ SI=ØØØ1
 DI=ØØØ6 CS=Ø1ØØ DS=Ø12A SS=Ø12A ES=ØØØØ IP=Ø1B5 FL=F2Ø2
.
```

The three matrices are displayed. Note that a word display has been specified by using the "DW" Command and that the addresses have been specified relative to the DS register. The addresses of XROW, YROW, and Z$ROW may be found in the debug map given by QRL86. Note that the values stored in the matrices are the same as those shown in Figures 8 and 9.

```
 .DW DS:10,4A
0010 0000 0000 0000 0000 0000 0001 0001 0001
0020 0001 0001 0002 0002 0002 0002 0002 0003
0030 0003 0003 0003 0003 0004 0004 0004 0004
0040 0004 0005 0005 0005 0005 0005
.DW DS:4C,68
004C 0000 FFFF
0050 FFFE 0000 FFFF FFFE 0000 FFFF FFFE 0000
0060 FFFF FFFE 0000 FFFF FFFE
.DW DS:6A,8C
006A 0000 0000 0000
0070 0000 FFFB FFF6 0000 FFF6 FFEC 0000 FFF1
0080 FFE2 0000 FFEC FFD8 0000 FFE7 FFCE
.
```

The "G" Command is used to reset the IP register to the start address of the program (∅∅2) and to specify a breakpoint at address ∅AEH, which is the address of statement 57 of the main program. Statement 57 is the point in the program after the X$ROW and Y$ROW matrices have been initialized, but before the matrix multiplication is performed. After the <CR> is typed, the program executes until the breakpoint is encountered. At this point, the monitor outputs a line specifying the number of the breakpoint, the CS and IP values and the first byte of the next instruction to be executed.

```
.G 01B5- 55 002,AE

BR1 @0100:00AE C7

.
```

Next, the single-step capability is used with the "N" command to execute single instructions. At any time, CPU registers may be examined or changed. In this example, the "X" command is used. Execution of succeeding instructions is caused by typing a comma (,).

```
 .N 00AE- C7 ,
 00B4- 81 ,
 00BA- 7E ,
 00BF- C7
.X
 AX=0018 BX=0018 CX=FFFE DX=0000 SP=00D0 BP=00D0 SI=0004
 DI=0006 CS=0100 DS=012A SS=012A ES=0000 IP=00BF FL=F293
.N 00BF- C7 ,
 00C5- 81 ,
 00CB- 7E
.
```

The contents of the X$ROW and Y$ROW matrices are examined and changed with the "SW" (substitute word) command. If a comma (,) is typed after the contents of memory are displayed, then the contents are left unchanged and the next word of memory is displayed. If a value followed by a comma or <CR> is entered, then the contents are changed. If a <CR> is entered, the substitute sequence is terminated.

```
.SW DS:1A, 0001- ,
001C 0001- ,
001E 0001- 10
.SW DS:5A, FFFF- ,
005C FFFE- ,
005E 0000- ,
0060 FFFF- 64
.
```

After the matrices are modified, execution is resumed with the "G" command. The max value is output and the INT 3 instruction executed. Finally, the contents of the 3 matrices are displayed.

```
 .G 00CB- 7E
MAX VALUE = +00480
@0100:01B5 55
.DW DS:10,8C
0010 0000 0000 0000 0000 0000 0001 0001 0010
0020 0001 0001 0002 0002 0002 0002 0002 0003
0030 0003 0003 0003 0003 0004 0004 0004 0004
0040 0004 0005 0005 0005 0005 0005 0000 FFFF
0050 FFFE 0000 FFFF FFFE 0000 FFFF FFFE 0000
0060 0064 FFFE 0000 FFFF FFFE 0000 0000 0000
0070 0000 0051 FFD8 0000 00C0 FFEC 0000 0120
0080 FFE2 0000 0180 FFD8 0000 01E0 FFCE
.
```

Expanding the Example Program's Memory Requirements

To illustrate how the iSBC 86/12 board may be used for executing 8086 programs which require large amounts of RAM, the example program will be modified. The matrix dimensions of the example will be changed from values of 6, 5 and 3 for the literal symbols of M, N, and P to values of 100, 50, 70. The three matrices will then be of size 100X50, 50X70, and 100X70. The memory required for these matrices is 15.5K words or 31K bytes. The data, constant, stack and memory segments which are contained in the group DGROUP will now comprise almost 32K bytes of memory.

The extra memory requirements will be supplied by using an iSBC 032 board with the iSBC 86/12 board in the iSBC 660 chassis. The iSBC 032 board is a 32K byte RAM board which is compatible with both 8- and 16-bit CPU boards. The base address of the board may be selected anywhere in a 0 to 1 megabyte range on any 16K byte boundary. 8- or 16-bit data transfers may be selected. The iSBC 032 board will be jumpered to respond to addresses in the 512K or 544K address space (20 bit hex address range to 8∅∅∅∅H to 87FFFH). This will illustrate the capabilities of the 8086 to access a 20-bit, 1 megabyte address range.

One other modification is required to the program. The magnitude of the numbers which would result from multiplying matrices of this size would greatly exceed the capacity of the 16-bit integer storage, even with the two matrices initialized to the small values they presently contain. To keep the example simple, the initialization values will be changed so all elements of the X$ROW matrix are set equal to 2 and all elements of the Y$ROW matrix are set equal to 3. The result of the multiplication should make all the elements of Z$ROW equal to 300.

The modified lines of program code are shown below.

```
                  /* MATRIX DIMENSIONS */
27    1        DECLARE M LITERALLY '1ØØ';
28    1        DECLARE N LITERALLY '5Ø';
29    1        DECLARE P LITERALLY '7Ø';


36    1        DO I = Ø TO (M-1);
37    2          DO J = Ø TO (N-1);
38    3            X$ROW(I).COL(J) = 2;
39    3          END;
40    2        END;


41    1        DO I = Ø TO (N-1);
42    2          DO J = Ø TO (P-1);
43    3            Y$ROW(I).COL(J) = 3;
44    3          END;
45    2        END;
```

The EXECUTION$VEHICLE module must be recompiled and then the three program modules must be linked and located using the QRL86 program. Specifying the SEGMENTS option of QRL86, the origin of the CODE segment which is in the group CGROUP is set at 1000H, as in the first example. However, the origin of the CONST, DATA STACK and MEMORY segments which make up the group DGROUP is set at 80000H.

```
QRL86 :F1:MATRIX.OBJ,:F1:FIND.OBJ,
SBCIOS.LIB SEGMENTS (CODE(1000H),
CONST (80000H), DATA STACK, MEMORY)
```

The memory map generated by QRL86 shows the CGROUP having a start address of 01000H and the DGROUP having a start address of 80000H.

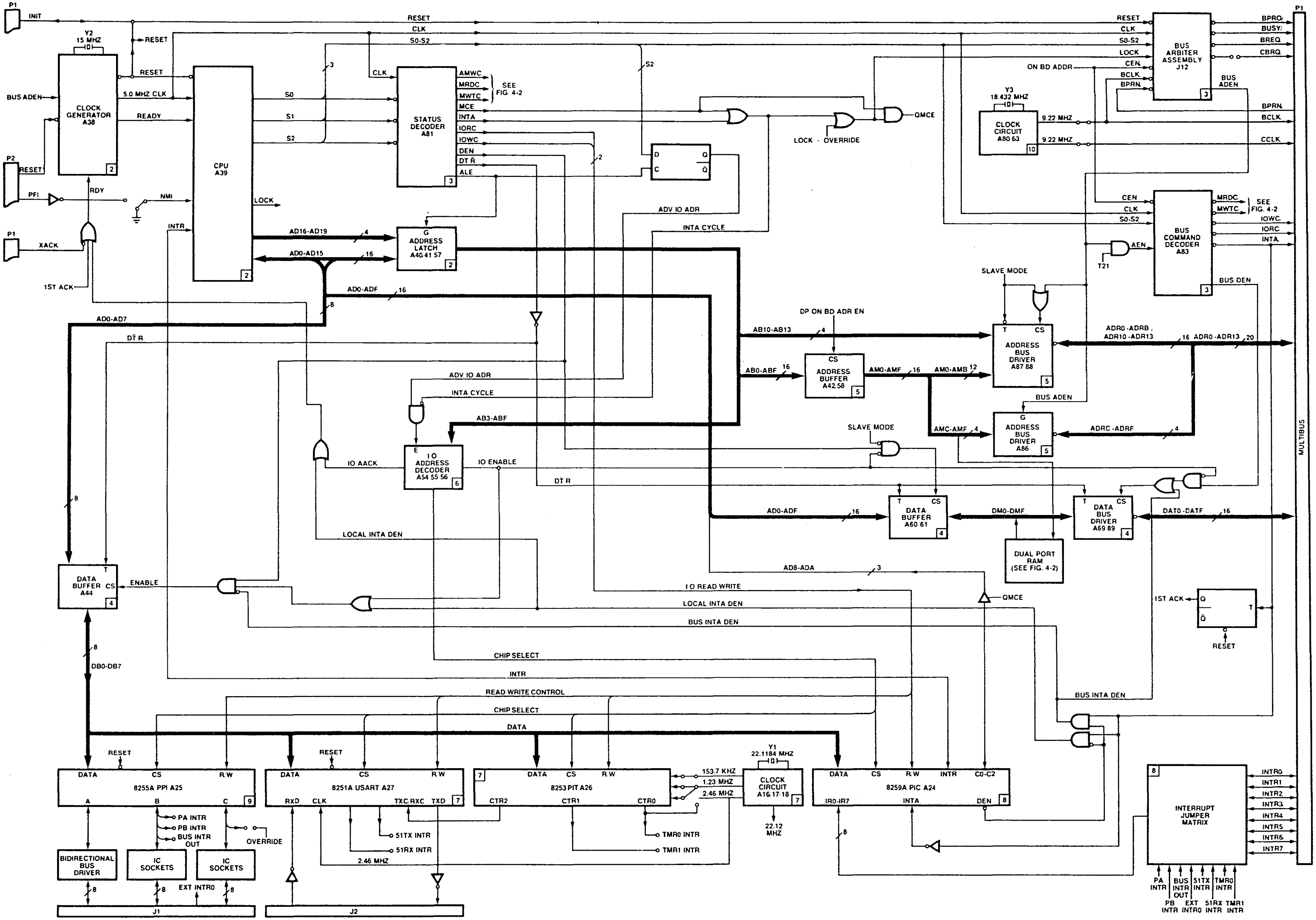
```
INVOKED BY:
QRL86 :F1:MATRIY.OBJ,:F1:FIND.OBJ,SBCIOS.LIB &
SEGMENTS(CODE(1ØØØH),CONST(8ØØØØH),DATA,STACK,MEMORY)

INPUT MODULES INCLUDED:
:F1:MATRIY.OBJ(EXECUTIONVEHICLE)
:F1:FIND.OBJ(FIND)
SBCIOS.LIB(SBCCO)

RESULT WRITTEN TO :F1:MATRIY(EXECUTIONVEHICLE)
START ADDRESS IS (Ø1ØØH,ØØØ2H)

 START   LTH  ALIGN NAME                       CLASS

Ø1ØØØH   298H   G   /GS/ CGROUP
Ø1ØØØH   21DH   W   CODE(EXECUTIONVEHICLE)   CODE
Ø121DH    41H   B   CODE(FIND)               CODE
Ø125EH    3AH   W   CODE(SBCCO)              CODE
                    /GE/ CGROUP
8ØØØØH  797ØH   G   /GS/ DGROUP
8ØØØØH    CH    W   CONST(EXECUTIONVEHICLE)  CONST
8ØØØCH    ØH    W   CONST(SBCCO)             CONST
8ØØØCH  792AH   W   DATA(EXECUTIONVEHICLE)   DATA
87936H    2H    W   DATA(FIND)               DATA
87938H    ØH    W   DATA(SBCCO)              DATA
8794ØH   3ØH   SW   STACK                    STACK
8797ØH    ØH    W   MEMORY                   MEMORY
                    /GE/ DGROUP
8797ØH    ØH    G   ??SEG(FIND)               (NULL)
```

The object code is then converted to hex format and downloaded to the iSBC 86/12 board. When the program is executed, the maximum value is calculated and output on the console.

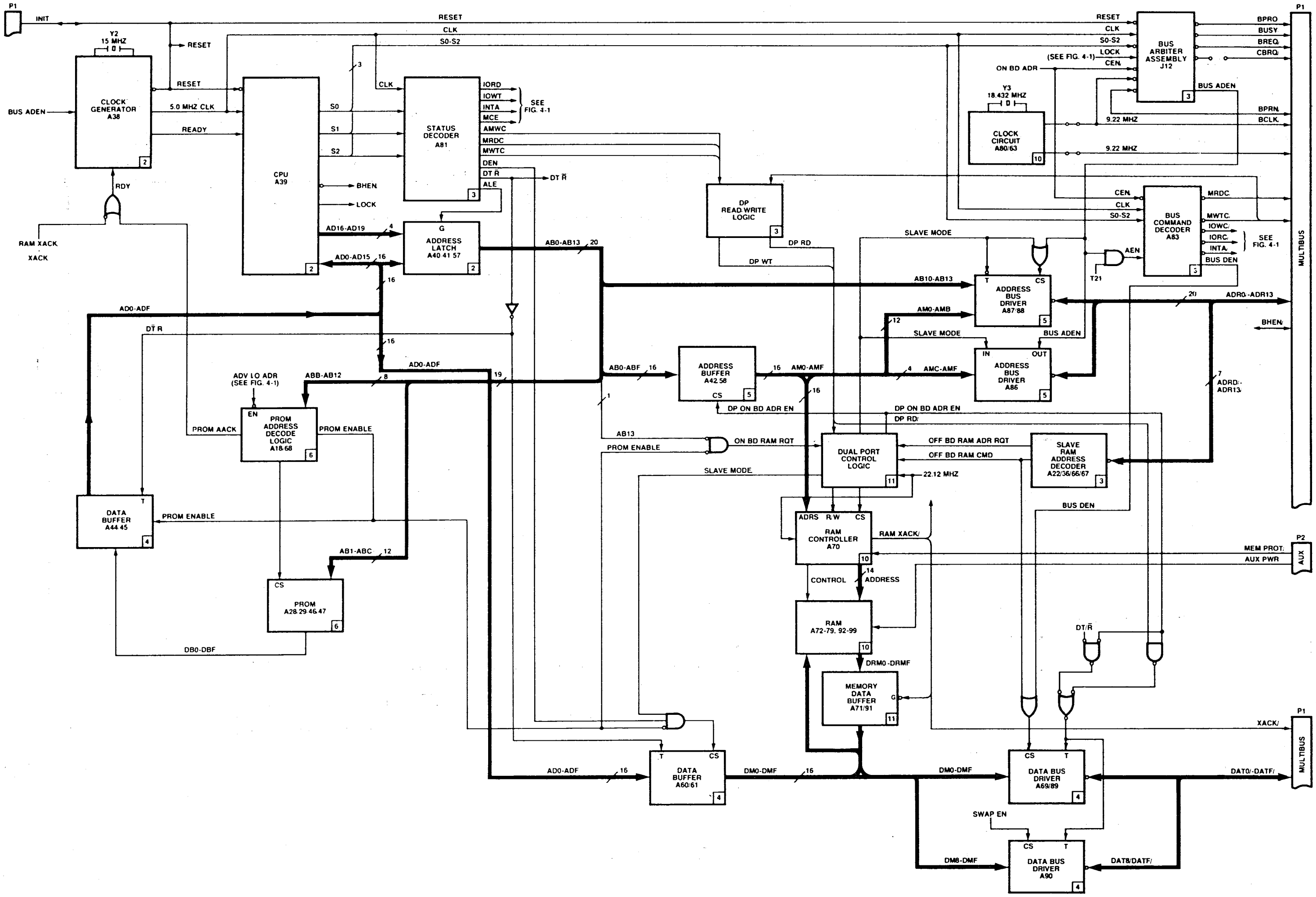
```
-SBC861

ISIS-II ISBC 86/12 LOADER, V1.2

ISBC 86/12 MONITOR,,V1.2
.LS,:F1:MATRIY.HEX

.S1AC, F4- CC
.G ØØØ2- FA
MAX VALUE = +ØØ3ØØ
@Ø1ØØ:Ø1AD 55
.
```

VI. CONCLUSION

This application note has described the iSBC 957 Intellec—iSBC 86/12 Interface and Execution Package, and how this package may be used to develop and debug programs for the 8086 processor. First, the iSBC 86/12 single board computer was described, followed by a detailed description of the iSBC 957 package and the iSBC 86/12 system monitor commands. The power and versatility of the iSBC 957 package and monitor commands for developing and debugging programs for the 8086 were illustrated by a program example. In the example a program which consisted of PL/M-86 and assembly language routines was presented. The program code was explained, and the steps required to compile, assemble, link, locate, and debug the program were illustrated. Finally, a typical debugging session using the iSBC 86/12 system monitor which illustrates the powerful capabilities of the monitor was presented.



iSBC™ 86/12 SIMPLIFIED LOGIC DIAGRAM
INPUT/OUTPUT AND INTERRUPT



iSBC™ 86/12 SIMPLIFIED LOGIC DIAGRAM
ROM/EPROM AND DUAL PORT RAM

APPENDIX B

PROGRAM LISTINGS FOR EXECUTION$VEHICLE AND FIND MODULES

```
PL/M-86 COMPILER    EXECUTIONVEHICLE

ISIS-II PL/M-86 V1.Ø COMPILATION OF MODULE EXECUTIONVEHICLE
OBJECT MODULE PLACED IN :F1:MATRIX.OBJ
COMPILER INVOKED BY:  PLM86 :F1:MATRIX.PLM DEBUG


                 /*      MATRIX MULTIPLICATION EXAMPLE PROGRAM

                     PL/M-86 MAIN PROGRAM WHICH:
                         A) INITIALIZES TWO INTEGER MATRICES
                         B) MULTIPLIES THE TWO MATRICES AND STORES THE RESULT IN A
                            THIRD MATRIX
                         C) CALLS AN ASSEMBLY LANGUAGE PROCEDURE WHICH SEARCHES THE
                            THIRD MATRIX FOR THE MAXIMUM VALUE
                         D) CALLS A PL/M PROCEDURE WHICH CONVERTS THE MAXIMUM VALUE
                            FROM INTEGER TO ASCII
                         E) CALLS A PROCEDURE WHICH OUTPUTS THE ASCII CHARACTERS ON
                            THE SYSTEM CONSOLE
                 */
    1            EXECUTION$VEHICLE:
                 DO;


                 /*  FIND$MX - EXTERNAL ASSEMBLY LANGUAGE PROCEDURE WHICH SEARCHES A
                         MATRIX FOR THE LARGEST ABSOLUTE MAGNITUDE.
                         PARAMETERS:
                            MATRIX$ADR -  ADDRESS OF THE MATRIX TO BE SEARCHED
                            ROWS - NUMBER OF ROWS IN THE MATRIX
                            COLS - NUMBER OF COLUMNS IN THE MATRIX
                 */
    2    1       FIND$MX: PROCEDURE (MATRIX$PTR, ROWS, COLS) INTEGER EXTERNAL;
    3    2       DECLARE (ROWS, COLS) INTEGER;
    4    2       DECLARE MATRIX$PTR POINTER;
    5    2       END FIND$MX;


                 /*  BIN$DEC$ASC - BINARY TO DECIMAL ASCII CONVERSION PROCEDURE
                         PARAMETERS:
                            VALUE - INTEGER VALUE TO BE CONVERTED TO ASCII
                            CHAR$ARRAY$ADR - ADDRESS OF 6 BYTE ARRAY WHERE ASCII
                                STRING CONTAINING THE VALUE WILL BE STORED
                 */
    6    1       BIN$DEC$ASC: PROCEDURE (VALUE, CHAR$ARRAY$ADR);

(A) 7    2       DECLARE (VALUE, TEMP, I) INTEGER;
    8    2       DECLARE CHAR$ARRAY$ADR POINTER;
    9    2       DECLARE (CHAR$ARRAY BASED CHAR$ARRAY$ADR) (6) BYTE;

   1Ø    2       IF VALUE < Ø THEN
   11    2       DO;
(G)12    3         CHAR$ARRAY(Ø) = '-';    /* SIGN CHARACTER */
   13    3         TEMP = -VALUE;
   14    3       END;
                 ELSE
   15    2       DO;
   16    3         CHAR$ARRAY(Ø) = '+';
   17    3         TEMP = VALUE;
   18    3       END;
   19    2       DO I = 5 TO 1 BY -1;
   2Ø    3         CHAR$ARRAY(I) = UNSIGN(TEMP MOD 1Ø) + 3ØH;
   21    3         TEMP = TEMP/1Ø;
                   /* ASCII CHARACTERS 3Ø THRU 39 HEX REPRESENT THE DIGITS Ø THRU 9. THUS
                     TO CONVERT AN INTEGER TO ASCII REPEATED DIVISIONS BY 1Ø AND ADDING
                     THE REMAINDER TO 3Ø HEX WILL ACCOMPLISH THE CONVERSION */
   22    3       END;

   23    2       END BIN$DEC$ASC;


                 /*  CO - EXTERNAL PROCEDURE TO OUTPUT A CHARACTER TO THE SYSTEM CONSOLE.
                       THIS PROCEDURE IS PART OF THE ISBC 957 LIBRARY FOR CONSOLE I/O
(M)                    PARAMETER:
                          CHAR - ASCII CHARACTER TO BE OUTPUT ON THE CONSOLE
                 */
   24    1       CO: PROCEDURE (CHAR) EXTERNAL;
   25    2       DECLARE CHAR BYTE;
   26    2       END CO;


                     /* MATRIX DIMENSIONS */
   27    1       DECLARE M LITERALLY '6';
   28    1       DECLARE N LITERALLY '5';
   29    1       DECLARE P LITERALLY '3';

                 /* THE THREE MATRICES ARE DECLARED AS ARRAYS OF STRUCTURES.  X$ROW IS COMPOSED
                    OF M STRUCTURES EACH OF WHICH IS COMPOSED OF N INTEGER ELEMENTS.  THUS
                    X$ROW MAY BE THOUGHT OF AS A M X N MATRIX.  THE MATRIX WILL BE STORED AS
                    A ROW-ORDER MATRIX WITH THE ELEMENTS OF EACH ROW STORED IN ADJACENT MEMORY
                    LOCATIONS.  Y$ROW IS DECLARED AS A N X P MATRIX AND Z$ROW AS A N X P MATRIX */
   3Ø    1       DECLARE X$ROW(M) STRUCTURE (COL(N) INTEGER);
   31    1       DECLARE Y$ROW(N) STRUCTURE (COL(P) INTEGER);
   32    1       DECLARE Z$ROW(M) STRUCTURE (COL(P) INTEGER);
   33    1       DECLARE (I,J,K,MAX) INTEGER;
   34    1       DECLARE MAX$ASC$ARRAY(6) BYTE;
   35    1       DECLARE TEXT(*) BYTE DATA ('MAX VALUE = ');
```

```
                /* INITIALIZE X$ROW SUCH THAT THE FIRST ROW IS SET EQUAL TO Ø, THE SECOND
                   ROW EQUAL TO 1, THE THIRD ROW EQUAL TO 2, ETC.  */
     36   1     DO I = Ø TO (M-1);
     37   2       DO J = Ø TO (N-1);
     38   3         X$ROW(I).COL(J) = I;
     39   3       END;
Ⓑ    40   2     END;

                /* INITIALIZE Y$ROW SUCH THAT THE FIRST COLUMN IS SET EQUAL TO Ø, THE
                   SECOND COLUMN EQUAL TO -1, AND THE THIRD COLUMN EQUAL TO -2.  */
     41   1     DO I = Ø TO (N-1);
     42   2       DO J = Ø TO (P-1);
     43   3         Y$ROW(I).COL(J) = -J;
     44   3       END;
     45   2     END;
                /* PERFORM MATRIX MULTIPLICATION  */
     46   1     DO K = Ø TO (P-1);
     47   2       DO I = Ø TO (M-1);
     48   3         Z$ROW(I).COL(K) = Ø;  /* SET Z$ROW ELEMENT TO Ø */
Ⓒ    49   3         DO J = Ø TO (N-1); /* SUM THE PRODUCT OF X$ROW ROW TERMS AND Y$ROW COLUMN TERMS */
     5Ø   4           Z$ROW(I).COL(K) = Z$ROW(I).COL(K) + ( X$ROW(I).COL(J) * Y$ROW(J).COL(K) );
     51   4         END;
     52   3       END;
     53   2     END;
Ⓓ    54   1     MAX = FIND$MX (@Z$ROW, M, P); /* FIND MAX VALUE OF Z$ROW */

Ⓔ    55   1     CALL BIN$DEC$ASC (MAX, @MAX$ASC$ARRAY); /* CONVERT TO DECIMAL ASCII */

     56   1     DO I = Ø TO (SIGNED(SIZE(TEXT)) - 1); /* OUTPUT HEADER TEXT */
     57   2       CALL CO(TEXT(I));
Ⓕ    58   2     END;

     59   1     DO I = Ø TO 5; /* OUTPUT ASCII MAX VALUE */
     6Ø   2       CALL CO(MAX$ASC$ARRAY(I));
     61   2     END;


     62   1     END EXECUTION$VEHICLE;


MODULE INFORMATION:

     CODE AREA SIZE     = Ø225H     549D
     CONSTANT AREA SIZE = ØØØCH      12D
     VARIABLE AREA SIZE = ØØ9ØH     144D
     MAXIMUM STACK SIZE = ØØØ8H       8D
     137 LINES READ
     Ø PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION

ISIS-II MCS-86 ASSEMBLER     ASSEMBLY OF MODULE FIND
OBJECT MODULE PLACED IN :F1:FIND.OBJ
ASSEMBLER INVOKED BY: ASM86 :F1:FIND.ASM DEBUG

LOC  OBJ

   LINE   SOURCE

      1          NAME    FIND
      2          PUBLIC  FINDMX
      3   ;
      4   ;
      5   ;
      6   ;                               FINDMX
      7   ;      ASSEMBLY LANGUAGE PROCEDURE TO FIND THE ELEMENT OF AN INTEGER
      8   ;      MATRIX WITH THE LARGEST ABSOLUTE MAGNITUDE.  THE VALUE OF THE
      9   ;      ELEMENT IS RETURNED IN THE AX REGISTER.
     1Ø   ;
     11   ;      PL/M CALLING SEQUENCE:
     12   ;              MAX$VALUE = FIND$MX(ADR$OF$MATRIX, #$OF$ROWS, #$OF$COLS);
     13   ;
     14   ;      PARAMETERS:
     15   ;              ADR$OF$MATRIX - ADDRESS OF THE MATRIX WHICH WILL BE SEARCHED
Ⓗ    16   ;              #$OF$ROWS - NUMBER OF ROWS IN THE MATRIX
     17   ;              #$OF$COLS - NUMBER OF COLUMNS IN THE MATRIX
     18   ;
     19   ;      PL/M WILL PASS THE THREE PARAMETERS IN THE CALL TO THIS PROCEDURE ON
     2Ø   ;      THE STACK.  ON ENTRY TO THE PROCEDURE SP+6 WILL POINT TO THE FIRST
     21   ;      PARAMETER(ADR$OF$MATRIX) AND SP+4 AND SP+2 WILL POINT TO THE SECOND
     22   ;      AND THIRD PARAMETERS.
     23   ;
     24   ;      THE PROCEDURE IS A TYPED PROCEDURE WHICH ASSIGNS THE MAXIMUM VALUE
     25   ;      IN THE MATRIX TO A VARIABLE (IN THIS CASE MAX$VALUE) IN A PL/M
     26   ;      ASSIGNMENT STATEMENT.  TO ACCOMPLISH THIS ASSIGNMENT THE VALUE IS
     27   ;      RETURNED IN THE AX REGISTER.
     28   ;
     29   ;
     3Ø   ;      THE ALGORITHM USED IS SIMILAR TO THE FOLLOWING PL/M CODE:
     31   ;              FOR I = Ø TO (#$OF$ROWS - 1);
     32   ;                FOR J = Ø TO (#$OF$COLS - 1);
     33   ;                  IF IABS(MATRIX(I).Y(J)) > IABS(MAX) THEN MAX = MATRIX(I).Y(J);
     34   ;                END;
     35   ;              END;
     36   ;
     37   ;              WHERE IABS(XYZ) REPRESENTS THE ABSOLUTE VALUE OF THE INTEGER XYZ
     38   ;
     39   ;
```

```
      LOC  OBJ                     LINE   SOURCE

                                    4Ø    ;
                                    41    ;       DEFINE GROUPS TO CONFORM WITH PL/M-86 CONVENTIONS. DATA, STACK, AND
                                    42    ;       CODE SEGMENTS WILL BE APPENDED TO THEIR RESPECTIVE SEGMENTS IN THE
                                    43    ;       PL/M-86 MODULES.
  I                                 44    DGROUP  GROUP   DATA,STACK
                                    45    CGROUP  GROUP   CODE
                                    46    ;
                                    47    ;       INSTRUCT THE ASSEMBLER THAT THE DS, SS, AND CS REGISTERS WILL CONTAIN
                                    48    ;       THE BASE ADDRESS VALUES FOR THE DGROUP, DGROUP AND CGROUP GROUPS.
                                    49            ASSUME  DS:DGROUP,SS:DGROUP,CS:CGROUP
                                    5Ø    ;
                                    51    ;
                                    52    ;
                                    53    ;***************DATA SEGMENT
  J                                 54    ;
      ----                          55    DATA    SEGMENT WORD PUBLIC 'DATA'
      ØØØØ ØØØØ                     56    MAX     DW      Ø
      ----                          57    DATA    ENDS
                                    58    ;
                                    59    ;***************STACK SEGMENT
                                    6Ø    ;
      ----                          61    STACK   SEGMENT STACK 'STACK'
      ØØØØ (14                      62            DW      14 DUP (Ø)      ;RESERVE 13 WORDS OF STACK FOR MONITOR
  K         ØØØØ
            )
                                    63                                    ;AND 1 WORD FOR FINDMX PROCEDURE
      ----                          64    STACK   ENDS
                                    65    ;
                                    66    ;***************CODE SEGMENT
                                    67    ;
      ----                          68    CODE    SEGMENT BYTE PUBLIC 'CODE'
                                    69    ;
                                    7Ø    ;PARAMETERS ON STACK, DISPLACEMENT FROM TOS INCREASED BY TWO DUE TO INITIAL PUSH
        ØØØ6[]                      71    NO_OF_ROWS      EQU     WORD PTR [BP+6]
        ØØØ4[]                      72    NO_OF_COLS      EQU     WORD PTR [BP+4]
        ØØØ8[]                      73    ADR_OF_MATRIX   EQU     WORD PTR [BP+8]
                                    74    ;
      ØØØØ                          75    FINDMX  PROC    NEAR            ;PROCEDURE DECLARATION
      ØØØØ 55                       76            PUSH    BP              ;SAVE BP REGISTER
      ØØØ1 8BEC                     77            MOV     BP,SP           ;BP POINTS TO PARAMETERS ON STACK
      ØØØ3 33D2                     78            XOR     DX,DX           ;SET DX = ABS OF CURRENT MAX = Ø
      ØØØ5 8BFA                     79            MOV     DI,DX           ;DI = I(ROW INDEX) = Ø
      ØØØ7 8BF2                     8Ø            MOV     SI,DX           ;SI = J(COLUMN INDEX) = Ø
      ØØØ9 8916ØØØØ         R       81            MOV     MAX,DX          ;MAX = CURRENT MAX = Ø
      ØØØD 8B4EØ4                   82            MOV     CX,NO_OF_COLS
      ØØ1Ø D1E1                     83            SHL     CX,1            ;CX = (#$OF$COLS) * 2
                                    84                                    ;TERMINATION FOR J(SI) INDEX
      ØØ12 8B5EØ8                   85            MOV     BX,ADR_OF_MATRIX        ;ADR$OF$MATRIX PARAMETER
                                    86                                    ;BX POINTS TO FIRST ELEMENT OF A GIVEN ROW
      ØØ15 8BØØ                     87    ABC:    MOV     AX,[BX][SI]     ;GET ELEMENT OF MATRIX
  L   ØØ17 ØBCØ                     88            OR      AX,AX           ;SET FLAGS
      ØØ19 79Ø2                     89            JNS     DEF             ;JUMP IF SIGN = Ø
      ØØ1B F7D8                     9Ø            NEG     AX              ;NEGATE TO FORM POSITIVE NUMBER
      ØØ1D 3BC2                     91    DEF:    CMP     AX,DX           ;COMPARE TO CURRENT MAX
      ØØ1F 7CØ7                     92            JL      XYZ             ;JUMP IF LESS THAN CURRENT MAX
      ØØ21 8BDØ                     93            MOV     DX,AX           ;MOVE TO ABS OF CURRENT MAX
      ØØ23 8BØØ                     94            MOV     AX,[BX][SI]     ;MOVE MATRIX VALUE TO CURRENT MAX
      ØØ25 A3ØØØØ           R       95            MOV     MAX,AX
      ØØ28 83C6Ø2                   96    XYZ:    ADD     SI,2            ;INCREMENT J INDEX BY TWO
      ØØ2B 3BF1                     97            CMP     SI,CX           ;END OF THIS ROW ??
      ØØ2D 72E6                     98            JB      ABC             ;IF NO, LOOP BACK FOR NEXT ELEMENT OF THIS ROW
      ØØ2F 8D18                     99            LEA     BX,[BX+SI]      ;BX = BX + (2 * #$OF$COLS), BX POINTS TO NEXT ROW
      ØØ31 BEØØØØ                  1ØØ            MOV     SI,Ø            ;J = Ø
      ØØ34 47                      1Ø1            INC     DI              ;I = I + 1
      ØØ35 3B7EØ6                  1Ø2            CMP     DI,NO OF ROWS   ;LAST ROW ??
      ØØ38 72DB                    1Ø3            JB      ABC             ;IF NO, DO THE NEXT ROW
      ØØ3A A1ØØØØ           R      1Ø4            MOV     AX,MAX          ;RETURN MAX VALUE IN AX REGISTER
      ØØ3D 5D                      1Ø5            POP     BP              ;RESTORE BP REGISTER
      ØØ3E C2Ø6ØØ                  1Ø6            RET     6               ;INCREMENT SP BY 6 AND RETURN TO CALLER
                                   1Ø7    ;
                                   1Ø8    FINDMX  ENDP
                                   1Ø9    ;
      ----                         11Ø    CODE    ENDS
                                   111    ;
                                   112            END


       SYMBOL TABLE LISTING
       ------ ----- -------


       NAME            TYPE      VALUE   ATTRIBUTES

       ??SEG . . . .   SEGMENT           SIZE=ØØØØH PARA PUBLIC
       ABC . . . . .   L NEAR    ØØ15H   CODE
       ADR_OF_MATRIX   V WORD    ØØØ8H   [BP]
       CGROUP. . . .   GROUP             CODE
       CODE. . . . .   SEGMENT           SIZE=ØØ41H BYTE   PUBLIC 'CODE'
       DATA. . . . .   SEGMENT           SIZE=ØØØ2H WORD   PUBLIC 'DATA'
       DEF . . . . .   L NEAR    ØØ1DH   CODE
       DGROUP. . . .   GROUP             DATA STACK
       FINDMX. . . .   L NEAR    ØØØØH   CODE PUBLIC
       MAX . . . . .   V WORD    ØØØØH   DATA
       NO_OF_COLS. .   V WORD    ØØØ4H   [BP]
       NO_OF_ROWS. .   V WORD    ØØØ6H   [BP]
       STACK . . . .   SEGMENT           SIZE=ØØ1CH PARA STACK 'STACK'
       XYZ . . . . .   L NEAR    ØØ28H   CODE

       ASSEMBLY COMPLETE, NO ERRORS FOUND
```

N

```
ISIS-II QRL-86, V1.1

INVOKED BY:
QRL86 :F1:MATRIX.OBJ,:F1:FIND.OBJ,SBCIOS.LIB ORIGIN(100ØH)

INPUT MODULES INCLUDED:
:F1:MATRIX.OBJ(EXECUTIONVEHICLE)
:F1:FIND.OBJ(FIND)
SBCIOS.LIB(SBCCO)

RESULT WRITTEN TO :F1:MATRIX(EXECUTIONVEHICLE)
START ADDRESS IS (Ø1ØØH,ØØØ2H)

 START   LTH   ALIGN NAME                          CLASS

Ø1ØØØH   2AØH    G    /GS/ CGROUP
Ø1ØØØH   225H    W    CODE(EXECUTIONVEHICLE)    CODE
Ø1225H    41H    B    CODE(FIND)                CODE
Ø1266H    3AH    W    CODE(SBCCO)               CODE
                      /GE/ CGROUP
Ø12AØH    DØH    G    /GS/ DGROUP
Ø12AØH     CH    W    CONST(EXECUTIONVEHICLE)   CONST
Ø12ACH     ØH    W    CONST(SBCCO)              CONST
Ø12ACH    9ØH    W    DATA(EXECUTIONVEHICLE)    DATA
Ø133CH     2H    W    DATA(FIND)                DATA
Ø133EH     ØH    W    DATA(SBCCO)               DATA
Ø134ØH    3ØH   SW    STACK                     STACK
Ø137ØH     ØH    W    MEMORY                    MEMORY
                      /GE/ DGROUP
Ø 37ØH     ØH    G    ??SEG(FIND)               (NULL)
```

P

```
DEBUG MAP OF :F1:MATRIX(EXECUTIONVEHICLE)

                MODULE: EXECUTIONVEHICLE   Ø1ØØH,Ø1E1H  LINE #: 19            Ø1ØØH,Ø139H  LINE #: 52
Ø12AH,ØØDØH  SYMBOL: MEMORY             Ø1ØØH,Ø1FBH  LINE #: 20            Ø1ØØH,Ø142H  LINE #: 53
Ø1ØØH,Ø1B5H  SYMBOL: BINDECASC          Ø1ØØH,Ø213H  LINE #: 21            Ø1ØØH,Ø14BH  LINE #: 54
Ø12AH,ØØØCH  SYMBOL: TEMP               Ø1ØØH,Ø21EH  LINE #: 22            Ø1ØØH,Ø15EH  LINE #: 55
Ø12AH,ØØØEH  SYMBOL: I                  Ø1ØØH,Ø221H  LINE #: 23            Ø1ØØH,Ø169H  LINE #: 56
Ø12AH,ØØ1ØH  SYMBOL: XROW               Ø1ØØH,ØØØ2H  LINE #: 36            Ø1ØØH,Ø17AH  LINE #: 57
Ø12AH,ØØ4CH  SYMBOL: YROW               Ø1ØØH,ØØ21H  LINE #: 37            Ø1ØØH,Ø185H  LINE #: 58
Ø12AH,ØØ6AH  SYMBOL: ZROW               Ø1ØØH,ØØ32H  LINE #: 38            Ø1ØØH,Ø18EH  LINE #: 59
Ø12AH,ØØ8EH  SYMBOL: I                  Ø1ØØH,ØØ4BH  LINE #: 39            Ø1ØØH,Ø19FH  LINE #: 6Ø
Ø12AH,ØØ9ØH  SYMBOL: J                  Ø1ØØH,ØØ54H  LINE #: 4Ø            Ø1ØØH,Ø1AAH  LINE #: 61
Ø12AH,ØØ92H  SYMBOL: K                  Ø1ØØH,ØØ5DH  LINE #: 41            Ø1ØØH,Ø1B3H  LINE #: 62
Ø12AH,ØØ94H  SYMBOL: MAX                Ø1ØØH,ØØ6EH  LINE #: 42                         MODULE: FIN
Ø12AH,ØØ96H  SYMBOL: MAXASCARRAY        Ø1ØØH,ØØ7FH  LINE #: 43            Ø1ØØH,Ø23AH  SYMBOL: ABC
Ø12AH,ØØØØH  SYMBOL: TEXT               Ø1ØØH,ØØ9CH  LINE #: 44            Ø1ØØH,Ø242H  SYMBOL: DEF
Ø1ØØH,Ø1B5H  LINE #: 6                  Ø1ØØH,ØØA5H  LINE #: 45            Ø1ØØH,Ø225H  SYMBOL: FINDMX
Ø1ØØH,Ø1B8H  LINE #: 1Ø                 Ø1ØØH,ØØAEH  LINE #: 46            Ø12AH,ØØ9CH  SYMBOL: MAX
Ø1ØØH,Ø1C2H  LINE #: 12                 Ø1ØØH,ØØBFH  LINE #: 47            Ø1ØØH,Ø24DH  SYMBOL: XYZ
Ø1ØØH,Ø1C8H  LINE #: 13                 Ø1ØØH,ØØDØH  LINE #: 48            Ø1ØØH,Ø225H  PUBLIC: FINDMX
Ø1ØØH,Ø1D1H  LINE #: 14                 Ø1ØØH,ØØE7H  LINE #: 49                         MODULE: SBCCO
Ø1ØØH,Ø1D4H  LINE #: 16                 Ø1ØØH,ØØF8H  LINE #: 5Ø            Ø1ØØH,Ø266H  PUBLIC: CO
Ø1ØØH,Ø1DAH  LINE #: 17                 Ø1ØØH,Ø13ØH  LINE #: 51
```

APPENDIX C

PROGRAM LISTING FOR EXECUTION$VEHICLE MODULE WITH CODE EXPANSION

```
PL/M-86 COMPILER    EXECUTIONVEHICLE


ISIS-II PL/M-86 V1.0 COMPILATION OF MODULE EXECUTIONVEHICLE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM86 :F1:MATRIX.PLM DEBUG CODE NOOBJECT PRINT(:F1:MATRIX.XLS)


             /*      MATRIX MULTIPLICATION EXAMPLE PROGRAM

                 PL/M-86 MAIN PROGRAM WHICH:
                     A) INITIALIZES TWO INTEGER MATRICES
                     B) MULTIPLIES THE TWO MATRICES AND STORES THE RESULT IN A
                        THIRD MATRIX
                     C) CALLS AN ASSEMBLY LANGUAGE PROCEDURE WHICH SEARCHES THE
                        THIRD MATRIX FOR THE MAXIMUM VALUE
                     D) CALLS A PL/M PROCEDURE WHICH CONVERTS THE MAXIMUM VALUE
                        FROM INTEGER TO ASCII
                     E) CALLS A PROCEDURE WHICH OUTPUTS THE ASCII CHARACTERS ON
                        THE SYSTEM CONSOLE
             */

  1          EXECUTION$VEHICLE:
             DO;


             /*  FIND$MX - EXTERNAL ASSEMBLY LANCUAGE PROCEDURE WHICH SEARCHES A
                     MATRIX FOR THE LARGEST ABSOLUTE MAGNITUDE.
                     PARAMETERS:
                         MATRIX$ADR -  ADDRESS OF THE MATRIX TO BE SEARCHED
                         ROWS - NUMBER OF ROWS IN THE MATRIX
                         COLS - NUMBER OF COLUMNS IN THE MATRIX
             */
  2   1      FIND$MX: PROCEDURE (MATRIX$PTR, ROWS, COLS) INTEGER EXTERNAL;
  3   2      DECLARE (ROWS, COLS) INTEGER;
  4   2      DECLARE MATRIX$PTR POINTER;
  5   2      END FIND$MX;


             /*  BIN$DEC$ASC - BINARY TO DECIMAL ASCII CONVERSION PROCEDURE
                     PARAMETERS:
                         VALUE - INTEGER VALUE TO BE CONVERTED TO ASCII
                         CHAR$ARRAY$ADR - ADDRESS OF 6 BYTE ARRAY WHERE ASCII
                             STRING CONTAINING THE VALUE WILL BE STORED
             */
  6   1      BIN$DEC$ASC: PROCEDURE (VALUE, CHAR$ARRAY$ADR);
                                                   ; STATEMENT # 6
                       BINDECASC         PROC NEAR
        01B5  55                  PUSH    BP
        01B6  8BEC                MOV     BP,SP

  7   2      DECLARE (VALUE, TEMP, I) INTEGER;
  8   2      DECLARE CHAR$ARRAY$ADR POINTER;
  9   2      DECLARE (CHAR$ARRAY BASED CHAR$ARRAY$ADR) (6) BYTE;

 10   2      IF VALUE < 0 THEN
                                                   ; STATEMENT # 10
        01B8  817E060000          CMP     [BP].VALUE,0H
        01BD  7C03                JL      $+5H
        01BF  E91200              JMP     @1
 11   2      DO;
 12   3        CHAR$ARRAY(0) = '-';    /* SIGN CHARACTER */
                                                   ; STATEMENT # 12
        01C2  8B5E04              MOV     BX,[BP].CHARARRAYADR
        01C5  C6072D              MOV     CHARARRAY[BX],2DH
 13   3        TEMP = -VALUE;
                                                   ; STATEMENT # 13
        01C8  8B4606              MOV     AX,[BP].VALUE
        01CB  F7D8                NEG     AX
        01CD  89060000            MOV     TEMP,AX
 14   3      END;
                                                   ; STATEMENT # 14
        01D1  E90D00              JMP     @2
                        @1:
             ELSE
 15   2      DO;
 16   3        CHAR$ARRAY(0) = '+';
                                                   ; STATEMENT # 16
        01D4  8B5E04              MOV     BX,[BP].CHARARRAYADR
        01D7  C6072B              MOV     CHARARRAY[BX],2BH
 17   3        TEMP = VALUE;
                                                   ; STATEMENT # 17
        01DA  8B4606              MOV     AX,[BP].VALUE
        01DD  89060000            MOV     TEMP,AX
 18   3      END;
                        @2:
 19   2      DO I = 5 TO 1 BY -1;
                                                   ; STATEMENT # 19
        01E1  C70602000500        MOV     I,5H
        01E7  E90600              JMP     @5
                        @3:
        01EA  81060200FFFF        ADD     I,0FFFFH
```

```
                       @5:
        Ø1FØ  813EØ2ØØØ1ØØ     CMP     I,1H
        Ø1F6  7DØ3             JGE     $+5H
        Ø1F8  E926ØØ           JMP     @4
2Ø   3        CHAR$ARRAY(I) = UNSIGN(TEMP MOD 1Ø) + 3ØH;
                                                ; STATEMENT # 2Ø
        Ø1FB  8BØ6ØØØØ         MOV     AX,TEMP
        Ø1FF  B9ØAØØ           MOV     CX,ØAH
        Ø2Ø2  31D2             XOR     DX,DX
        Ø2Ø4  F7F9             IDIV    CX
        Ø2Ø6  81C23ØØØ         ADD     DX,3ØH
        Ø2ØA  8B5EØ4           MOV     BX,[BP].CHARARRAYADR
        Ø2ØD  8B36Ø2ØØ         MOV     SI,I
        Ø211  881Ø             MOV     [BX].CHARARRAY[SI],DL
21   3        TEMP = TEMP/1Ø;
                                                ; STATEMENT # 21
              /* ASCII CHARACTERS 3Ø THRU 39 HEX REPRESENT THE DIGITS Ø THRU 9. THUS
                TO CONVERT AN INTEGER TO ASCII REPEATED DIVISIONS BY 1Ø AND ADDING
                THE REMAINDER TO 3Ø HEX WILL ACCOMPLISH THE CONVERSION */
        Ø213  8BØ6ØØØØ         MOV     AX,TEMP
        Ø217  99               CWD
        Ø218  F7F9             IDIV    CX
        Ø21A  89Ø6ØØØØ         MOV     TEMP,AX
22   3      END;
                                                ; STATEMENT # 22
        Ø21E  E9C9FF           JMP     @3
                       @4:

23   2      END BIN$DEC$ASC;
                                                ; STATEMENT # 23
        Ø221  5D               POP     BP
        Ø222  C2Ø4ØØ           RET     4H
                      BINDECASC      ENDP


            /*  CO - EXTERNAL PROCEDURE TO OUTPUT A CHARACTER TO THE SYSTEM CONSOLE.
                  THIS PROCEDURE IS PART OF THE ISBC 957 LIBRARY FOR CONSOLE I/O
                  PARAMETER:
                    CHAR - ASCII CHARACTER TO BE OUTPUT ON THE CONSOLE
            */
24   1      CO: PROCEDURE (CHAR) EXTERNAL;
25   2      DECLARE CHAR BYTE;
26   2      END CO;


               /* MATRIX DIMENSIONS */
27   1      DECLARE M LITERALLY '6';
28   1      DECLARE N LITERALLY '5';
29   1      DECLARE P LITERALLY '3';

            /* THE THREE MATRICES ARE DECLARED AS ARRAYS OF STRUCTURES.  X$ROW IS COMPOSED
               OF M STRUCTURES EACH OF WHICH IS COMPOSED OF N INTEGER ELEMENTS.  THUS
               X$ROW MAY BE THOUGHT OF AS A M X N MATRIX.  THE MATRIX WILL BE STORED AS
               A ROW-ORDER MATRIX WITH THE ELEMENTS OF EACH ROW STORED IN ADJACENT MEMORY
               LOCATIONS.  Y$ROW IS DECLARED AS A N X P MATRIX AND Z$ROW AS A N X P MATRIX */
3Ø   1      DECLARE X$ROW(M) STRUCTURE (COL(N) INTEGER);
31   1      DECLARE Y$ROW(N) STRUCTURE (COL(P) INTEGER);
32   1      DECLARE Z$ROW(M) STRUCTURE (COL(P) INTEGER);

33   1      DECLARE (I,J,K,MAX) INTEGER;
34   1      DECLARE MAX$ASC$ARRAY(6) BYTE;
35   1      DECLARE TEXT(*) BYTE DATA ('MAX VALUE = ');


            /* INITIALIZE X$ROW SUCH THAT THE FIRST ROW IS SET EQUAL TO Ø, THE SECOND
               ROW EQUAL TO 1, THE THIRD ROW EQUAL TO 2, ETC.  */
36   1      DO I = Ø TO (M-1);
                                                ; STATEMENT # 36
        ØØØ2  FA               CLI
        ØØØ3  2E8E16ØØØØ       MOV     SS,CS:@@STACK$FRAME
        ØØØ8  BCØ8ØØ           MOV     SP,@@STACK$OFFSET
        ØØØB  8BEC             MOV     BP,SP
        ØØØD  16               PUSH    SS
        ØØØE  1F               POP     DS
        ØØØF  FB               STI
        ØØ1Ø  C7Ø682ØØØØØØ     MOV     I,ØH
                       @6:
        ØØ16  813E82ØØØ5ØØ     CMP     I,5H
        ØØ1C  7EØ3             JLE     $+5H
        ØØ1E  E93CØØ           JMP     @7
37   2        DO J = Ø TO (N-1);
                                                ; STATEMENT # 37
        ØØ21  C7Ø684ØØØØØØ     MOV     J,ØH
                       @8:
        ØØ27  813E84ØØØ4ØØ     CMP     J,4H
        ØØ2D  7EØ3             JLE     $+5H
        ØØ2F  E922ØØ           JMP     @9
38   3          X$ROW(I).COL(J) = I;
                                                ; STATEMENT # 38
        ØØ32  8BØ682ØØ         MOV     AX,I
        ØØ36  B9ØAØØ           MOV     CX,ØAH
        ØØ39  F7E9             IMUL    CX
        ØØ3B  8B3684ØØ         MOV     SI,J
        ØØ3F  D1E6             SHL     SI,1
        ØØ41  89C3             MOV     BX,AX
        ØØ43  8BØE82ØØ         MOV     CX,I
        ØØ47  89884ØØØ         MOV     [BX].XROW[SI],CX
39   3        END;
```

```
                                                          ; STATEMENT # 39
          004B  81068400010Ø     ADD        J,1H
          ØØ51  E9D3FF           JMP        @8
                          @9:
 40   2      END;
                                                          ; STATEMENT # 40
          ØØ54  8106820001ØØ     ADD        I,1H
          ØØ5A  E9B9FF           JMP        @6
                          @7:

             /* INITIALIZE Y$ROW SUCH THAT THE FIRST COLUMN IS SET EQUAL TO Ø, THE
                SECOND COLUMN EQUAL TO -1, AND THE THIRD COLUMN EQUAL TO -2.  */
 41   1      DO I = Ø TO (N-1);
                                                          ; STATEMENT # 41
          ØØ5D  C7Ø682ØØØØØØ     MOV        I,ØH
                          @1Ø:
          ØØ63  813E82ØØØ4ØØ     CMP        I,4H
          ØØ69  7EØ3             JLE        $+5H
          ØØ6B  E94ØØØ           JMP        @11
 42   2        DO J = Ø TO (P-1);
                                                          ; STATEMENT # 42
          ØØ6E  C7Ø684ØØØØØØ     MOV        J,ØH
                          @12:
          ØØ74  813E84ØØØ2ØØ     CMP        J,2H
          ØØ7A  7EØ3             JLE        $+5H
          ØØ7C  E926ØØ           JMP        @13
 43   3          Y$ROW(I).COL(J) = -J;
                                                          ; STATEMENT # 43
          ØØ7F  8BØ684ØØ         MOV        AX,J
          ØØ83  F7D8             NEG        AX
          ØØ85  5Ø               PUSH       AX        ; 1
          ØØ86  8BØ682ØØ         MOV        AX,I
          ØØ8A  B9Ø6ØØ           MOV        CX,6H
          ØØ8D  F7E9             IMUL       CX
          ØØ8F  8B3684ØØ         MOV        SI,J
          ØØ93  D1E6             SHL        SI,1
          ØØ95  89C3             MOV        BX,AX
          ØØ97  59               POP        CX        ; 1
          ØØ98  89884ØØØ         MOV        [BX].YROW[SI],CX
 44   3        END;
                                                          ; STATEMENT # 44
          ØØ9C  81Ø684ØØØ1ØØ     ADD        J,1H
          ØØA2  E9CFFF           JMP        @12
                          @13:
 45   2      END;
                                                          ; STATEMENT # 45
          ØØA5  81Ø682ØØØ1ØØ     ADD        I,1H
          ØØAB  E9B5FF           JMP        @1Ø
                          @11:
             /* PERFORM MATRIX MULTIPLICATION  */
 46   1      DO K = Ø TO (P-1);
                                                          ; STATEMENT # 46
          ØØAE  C7Ø686ØØØØØØ     MOV        K,ØH
                          @14:
          ØØB4  813E86ØØØ2ØØ     CMP        K,2H
          ØØBA  7EØ3             JLE        $+5H
          ØØBC  E98CØØ           JMP        @15
 47   2        DO I = Ø TO (M-1);
                                                          ; STATEMENT # 47
          ØØBF  C7Ø682ØØØØØØ     MOV        I,ØH
                          @16:
          ØØC5  813E82ØØØ5ØØ     CMP        I,5H
          ØØCB  7EØ3             JLE        $+5H
          ØØCD  E972ØØ           JMP        @17
 48   3          Z$ROW(I).COL(K) = Ø;  /* SET Z$ROW ELEMENT TO Ø */
                                                          ; STATEMENT # 48
          ØØDØ  8BØ682ØØ         MOV        AX,I
          ØØD4  B9Ø6ØØ           MOV        CX,6H
          ØØD7  F7E9             IMUL       CX
          ØØD9  8B3686ØØ         MOV        SI,K
          ØØDD  D1E6             SHL        SI,1
          ØØDF  89C3             MOV        BX,AX
          ØØE1  C7Ø5EØØØØØ       MOV        [BX].ZROW[SI],ØH
 49   3          DO J = Ø TO (N-1); /* SUM THE PRODUCT OF X$ROW ROW TERMS AND Y$ROW COLUMN TERMS */
                                                          ; STATEMENT # 49
          ØØE7  C7Ø684ØØØØØØ     MOV        J,ØH
                          @18:
          ØØED  813E84ØØØ4ØØ     CMP        J,4H
          ØØF3  7EØ3             JLE        $+5H
          ØØF5  E941ØØ           JMP        @19
 5Ø   4            Z$ROW(I).COL(K) = Z$ROW(I).COL(K) + ( X$ROW(I).COL(J) * Y$ROW(J).COL(K) );
                                                          ; STATEMENT # 5Ø
          ØØF8  8BØ682ØØ         MOV        AX,I
          ØØFC  B9ØAØØ           MOV        CX,ØAH
          ØØFF  F7E9             IMUL       CX
          Ø1Ø1  8B3684ØØ         MOV        SI,J
          Ø1Ø5  D1E6             SHL        SI,1
          Ø1Ø7  5Ø               PUSH       AX        ; 1
          Ø1Ø8  8BØ684ØØ         MOV        AX,J
          Ø1ØC  B9Ø6ØØ           MOV        CX,6H
          Ø1ØF  F7E9             IMUL       CX
          Ø111  8B3E86ØØ         MOV        DI,K
          Ø115  D1E7             SHL        DI,1
          Ø117  89C3             MOV        BX,AX
          Ø119  8B814ØØØ         MOV        AX,[BX].YROW[DI]
          Ø11D  5B               POP        BX        ; 1
          Ø11E  F7A8Ø4ØØ         IMUL       [BX].XROW[SI]
          Ø122  5Ø               PUSH       AX        ; 1
          Ø123  8BØ682ØØ         MOV        AX,I
          Ø127  F7E9             IMUL       CX
          Ø129  89C3             MOV        BX,AX
```

```
           Ø12B  58              POP     AX       ; 1
           Ø12C  Ø1815EØØ        ADD     [BX].ZROW[DI],AX
 51   4            END;
                                                  ; STATEMENT # 51
           Ø13Ø  81Ø684ØØØ1ØØ    ADD     J,1H
           Ø136  E9B4FF          JMP     @18
                         @19:
 52   3          END;
                                                  ; STATEMENT # 52
           Ø139  81Ø682ØØØ1ØØ    ADD     I,1H
           Ø13F  E983FF          JMP     @16
                         @17:
 53   2        END;
                                                  ; STATEMENT # 53
           Ø142  81Ø686ØØØ1ØØ    ADD     K,1H
           Ø148  E959FF          JMP     @14
                         @15:

 54   1        MAX = FINDSMX (@Z$ROW, M, P); /* FIND MAX VALUE OF Z$ROW */
                                                  ; STATEMENT # 54

           Ø14B  B85EØØ          MOV     AX,OFFSET(ZROW)
           Ø14E  5Ø              PUSH    AX       ; 1
           Ø14F  B8Ø6ØØ          MOV     AX,6H
           Ø152  5Ø              PUSH    AX       ; 2
           Ø153  B8Ø3ØØ          MOV     AX,3H
           Ø156  5Ø              PUSH    AX       ; 3
           Ø157  E8ØØØØ          CALL    FINDMX
           Ø15A  89Ø688ØØ        MOV     MAX,AX
 55   1        CALL BIN$DEC$ASC (MAX, @MAX$ASC$ARRAY); /* CONVERT TO DECIMAL ASCII */
                                                  ; STATEMENT # 55
           Ø15E  FF3688ØØ        PUSH    MAX      ; 1
           Ø162  B88AØØ          MOV     AX,OFFSET(MAXASCARRAY)
           Ø165  5Ø              PUSH    AX       ; 2
           Ø166  E84CØØ          CALL    BINDECASC

 56   1        DO I = Ø TO (SIGNED(SIZE(TEXT)) - 1); /* OUTPUT HEADER TEXT */
                                                  ; STATEMENT # 56
           Ø169  C7Ø682ØØØØØØ    MOV     I,ØH
                         @2Ø:
           Ø16F  813E82ØØØBØØ    CMP     I,ØBH
           Ø175  7EØ3            JLE     $+5H
           Ø177  E914ØØ          JMP     @21
 57   2          CALL CO(TEXT(I));
                                                  ; STATEMENT # 57
           Ø17A  8B1E82ØØ        MOV     BX,I
           Ø17E  FFB7ØØØØ        PUSH    TEXT[BX]; 1
           Ø182  E8ØØØØ          CALL    CO
 58   2        END;
                                                  ; STATEMENT # 58
           Ø185  81Ø682ØØØ1ØØ    ADD     I,1H
           Ø18B  E9E1FF          JMP     @2Ø
                         @21:

 59   1        DO I = Ø TO 5; /* OUTPUT ASCII MAX VALUE */
                                                  ; STATEMENT # 59
           Ø18E  C7Ø682ØØØØØØ    MOV     I,ØH
                         @22:
           Ø194  813E82ØØØ5ØØ    CMP     I,5H
           Ø19A  7EØ3            JLE     $+5H
           Ø19C  E914ØØ          JMP     @23
 6Ø   2          CALL CO(MAX$ASC$ARRAY(I));
                                                  ; STATEMENT # 6Ø
           Ø19F  8B1E82ØØ        MOV     BX,I
           Ø1A3  FFB78AØØ        PUSH    MAXASCARRAY[BX]; 1
           Ø1A7  E8ØØØØ          CALL    CO
 61   2        END;
                                                  ; STATEMENT # 61
           Ø1AA  81Ø682ØØØ1ØØ    ADD     I,1H
           Ø1BØ  E9E1FF          JMP     @22
                         @23:

 62   1        END EXECUTION$VEHICLE;
                                                  ; STATEMENT # 62
           Ø1B3  FB              STI
           Ø1B4  F4              HLT



MODULE INFORMATION:

     CODE AREA SIZE     = Ø225H    549D
     CONSTANT AREA SIZE = ØØØCH     12D
     VARIABLE AREA SIZE = ØØ9ØH    144D
     MAXIMUM STACK SIZE = ØØØ8H      8D
     137 LINES READ
     Ø PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```