

**APPENDIX A (Continued)**  
**P2 CONNECTOR PIN ASSIGNMENT OF OPTIONAL BUS SIGNALS**

PIN	(COMPONENT SIDE)		PIN	(CIRCUIT SIDE)	
	MNEMONIC	DESCRIPTION		MNEMONIC	DESCRIPTION
1	GND	Signal GND	2	GND	Signal GND
3	5 VB	+5V Battery	4	5 VB	+5V Battery
5		Reserved	6	VCCPP	+5V Pulsed Power
7	-5 VB	-5V Battery	8	-5 VB	-5V Battery
9		Reserved	10	Reserved	
11	12 VB	+12V Battery	12	12 VB	+12V Battery
13	PFSR/	Power Fail Sense Reset	14	Reserved	
15	-12 VB	-12V Battery	16	-12 VB	-12V Battery
17	PFSN/	Power Fail Sense	18	ACLO	AC Low
19	PFIN/	Power Fail Interrupt	20	MPRO/	Memory Protect
21	GND	Signal GND	22	GND	Signal GND
23	+15V	+15V	24	+15V	+15V
25	-15V	-15V	26	-15V	-15V
27	PAR1/	Parity 1	28	HALT/	Bus Master HALT
29	PAR2/	Parity 2	30	WAIT/	Bus Master WAIT STATE
31			32	ALE	Bus Master ALE
33	Reserved		34	Reserved	
35			36	Reserved	
37			38	AUX RESET/	Reset switch
39			40		
40			42	Reserved	
43			44		
45			46		
47			48		
49			50		
51			52		
53			54		
55			56		
57			58		
59			60		

**Notes:**

1. PFIN, on slave modules, if possible, should have the option of connecting to INT0/ on P1.
2. All undefined pins are reserved for future use.

All Mnemonics © Intel Corporation 1978

**APPENDIX B**  
**BUS TIMING SPECIFICATIONS SUMMARY**

Parameter	Description	Minimum	Maximum	Units
t <sub>BCY</sub>	Bus Clock Period	100	D.C.	ns
t <sub>BW</sub>	Bus Clock Width	0.35 t <sub>BCY</sub>	0.65 t <sub>BCY</sub>	
t <sub>SKEW</sub>	BCLK / skew		3	ns
t <sub>PD</sub>	Standard Bus Propagation Delay		3	
t <sub>AS</sub>	Address Set-Up Time (at Slave Board)	50		ns
t <sub>DS</sub>	Write Data Set-Up Time	50		ns
t <sub>AH</sub>	Address Hold Time	50		ns
t <sub>DHW</sub>	Write Data Hold Time	50		ns
t <sub>DXL</sub>	Read Data Set-Up Time To XACK	0		ns
t <sub>DHR</sub>	Read Data Hold Time	0	65	ns
t <sub>XAH</sub>	Acknowledge Hold Time	0	65	ns
t <sub>XACK</sub>	Acknowledge Time	0	t <sub>TOUT</sub>	ns
t <sub>CMD</sub>	Command Pulse Width	100	t <sub>TOUT</sub>	ns
t <sub>ID</sub>	Inhibit Delay	0	100 (Recommend < 100 ns)	ns
t <sub>XACKA</sub>	Acknowledge Time of of an Inhibited Slave	t <sub>IAD</sub> + 50 ns	t <sub>TOUT</sub>	
t <sub>XACKB</sub>	Acknowledge Time of an Inhibiting Slave	1.5	t <sub>TOUT</sub>	μs
t <sub>IAD</sub>	Acknowledge Disable from Inhibit (An internal parameter on an inhibited slave; used to determine t <sub>XACKA</sub> Min.)	0	100 (arbitrary)	ns
t <sub>AIZ</sub>	Address to Inhibits High delay		100	ns
t <sub>INTA</sub>	INTA / Width	250		ns
t <sub>CSEP</sub>	Command Separation	100		ns

**APPENDIX B (Continued)**  
**BUS TIMING SPECIFICATIONS SUMMARY**

Parameter	Description	Minimum	Maximum	Units
tBREQL	↓BCLK / to BREQ / Low Delay	0	35	ns
tBREQH	↓BCLK / to BREQ / High Delay	0	35	ns
tBPRNS	BPRN / to ↓BCLK / Setup Time	22		ns
tBUSY	BUSY / delay from ↓BCLK /	0	70	ns
tBUSYS	BUSY / to ↓BCLK / Setup Time	25		ns
tBPRO	↓BCLK / to BPRO / (CLK to Priority Out)	0	40	ns
tBPRNO	BPRN / to BPRO / (Priority In to Out)	0	30	ns
tCBRO	↓BCLK / to CBRQ / (CLK to Common Bus Request)	0	60	ns
tCBRQS	CBRQ / to ↓BCLK / Setup Time	35		ns
tCPM	Central Priority Module Resolution Delay (Parallel Priority)	0	tBCY - tBREQ - 2tPD - tBPRNS - tSKEW	
tCCY	C-clock Period	100	110	ns
tCW	C-clock Width	0.35 tCCY	0.65 tCCY	ns
tINIT	INIT / Width	5		ms
tINITS	INIT / to MPRO / Setup Time	100		ns
tPBD	Power Backup Logic Delay	0	200	ns
tPFINW	PFIN / Width	2.5		ms
tMPRO	MPRO / Delay	2.0	2.5	ms
tACLOW	ACLO / Width	3.0		ms
tPFSRW	PFSR / Width	100		ns
tTOUT	Timeout Delay	5	∞	ms
tDCH	D.C. Power Supply Hold from ALCO /	3.0		ms
tDCS	D.C. Power Supply Setup to ACLO /	5		ms

**APPENDIX C**  
**BUS DRIVERS, RECEIVERS, AND TERMINATIONS**

Driver 1,3						Receiver 2,3				Termination		
Bus Signals	Location	Type	IOL Min <sub>ma</sub>	IOH Min <sub>µa</sub>	CO Max <sub>pf</sub>	Location	IIL Max <sub>ma</sub>	IIH Max <sub>µa</sub>	CI Max <sub>pf</sub>	Location	Type	R Units
DAT0/-DATF/ (16 lines)	Masters and Slaves	TRI	16	-2000	300	Masters and Slaves	-0.8	125	18	1 place	Pullup	2.2 KΩ
ADRO/-ADRB/ BHEN/ (21 lines)	Masters	TRI	16	-2000	300	Slaves	-0.8	125	18	1 place	Pullup	2.2 KΩ
MRDC/,MWTC/	Masters	TRI	32	-2000	300	Slaves (Memory; memory- mapped I/O)	-2	125	18	1 place	Pullup	1 KΩ
IORC/,IOWC/	Masters	TRI	32	-2000	300	Slaves (I/O)	-2	125	18	1 place	Pullup	1 KΩ
XACK/	Slaves	TRI	32	-2000	300	Masters	-2	125	18	1 place	Pullup	510 Ω
INH1/,INH2/	Inhibiting Slaves	OC	16	—	300	Inhibited Slaves (RAM, PROM, ROM, Memory- Mapped I/O)	-2	50	18	1 place	Pullup	1 KΩ
BCLK/	1 place (Master us)	TTL	48	-3000	300	Master	-2	125	18	Mother- board	To +5V To GND	220 Ω 330 Ω
BREQ/	Each Master	TTL	5	-400	60	Central Priority Module	2	50	18	Central Priority Module (not req)	Pullup	1 KΩ
BPRO/	Each Master	TTL	5	-400	60	Next Master in Serial Priority Chain at its BPRN/	-1.6	50	18	(not req)		
BPRN/	Parallel: Central Priority Module Serial: Prev Masters BPRO/	TTL	5	-400	300	Master	-2	50		(not req)		
BUSY/, CBRQ	All Masters	O.C.	32	—	300	All Masters	-2	50	18	1 place	Pullup	1 KΩ
INIT/	Master	O.C.	32	—	300	All	-2	50	18	1 place	Pullup	2.2 KΩ
CCLK/	1 place	TTL	48	-3000	300	Any	-2	125	18	Mother- board	To +5V To GND	220 Ω 330 Ω
INTA/	Masters	TRI	32	-2000	300	Slaves (Interrupting I/O)	-2	125	18	1 place	Pullup	1 KΩ
INT0/-INT7/ (8 lines)	Slaves	O.C.	16	—	300	Masters	-1.6	40	18	1 place	Pullup	1 KΩ
PFSR/	User's Fron Panel?	TTL	16	-400	300	Slaves, Masters	-1.6	40	18	1 place	Pullup	1 KΩ
PFSN/	Power Back Up Unit	TTL	16	-400	300	Masters	-1.6	40	16	1 place	Pullup	1 KΩ
ACLO	Power Supply	O.C.	16	-400	300	Slaves, Masters	-1.6	40	18	1 place	Pullup	1 KΩ
PFIN/	Power Back- Up Unit	O.C.	16	-400	300	Masters	-1.6	40	18	1 place	Pullup	1 KΩ
MPRO/	Power Back- Up Unit	TTL	16	-400	300	Slaves Masters	-1.6	40	18	1 place	Pullup	1 KΩ

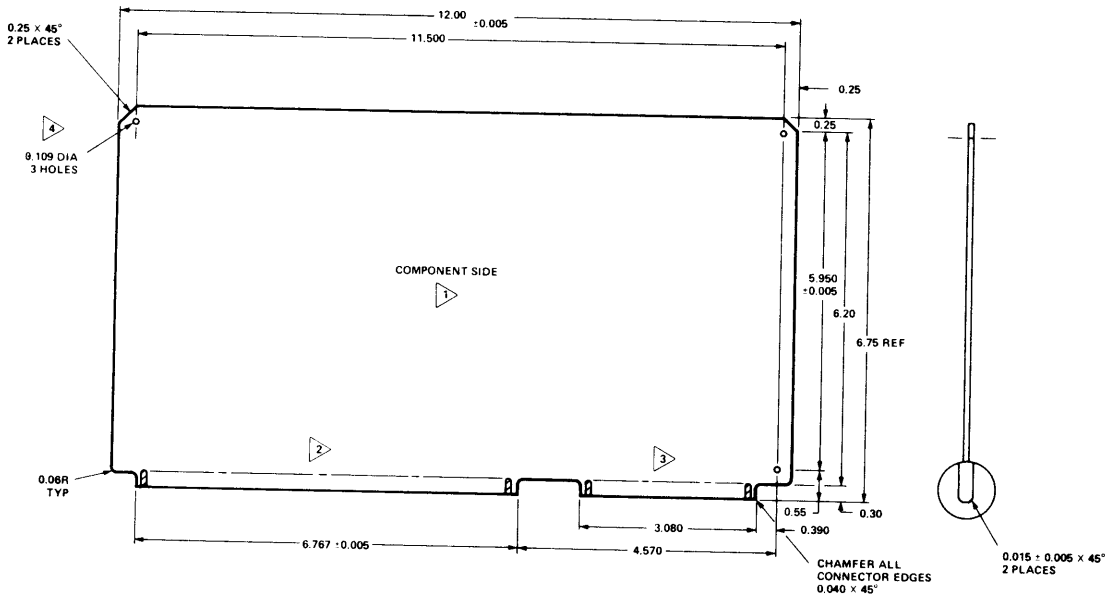
**APPENDIX C (Continued)**  
**BUS DRIVERS, RECEIVERS, AND TERMINATIONS**

Driver 1,3						Receiver 2,3				Termination		
Bus Signals	Location	Type	I <sub>OL</sub> Min <sub>ma</sub>	I <sub>OH</sub> Min <sub>μa</sub>	C <sub>O</sub> Max <sub>pf</sub>	Location	I <sub>IL</sub> Max <sub>ma</sub>	I <sub>IH</sub> Max <sub>μa</sub>	C <sub>I</sub> Max <sub>pf</sub>	Location	Type	R Units
Aux Reset/	User's Front Panel?	Switch to GND	—	—	—	Masters	-2	50	18	None		
Notes: 1. Driver Requirements I <sub>OH</sub> = High Output Current Drive I <sub>OL</sub> = Low Output Current Drive C <sub>O</sub> = Capacitance Drive Capability TRI = 3-State Drive O.C. = Open Collector Driver TTL = Totem-pole Driver 2. Receiver Requirements I <sub>IH</sub> = High Input Current Load I <sub>IL</sub> = Low Input Current Load C <sub>I</sub> = Capacitive Load 3. TTL low state must be $\geq -0.5v$ but $\leq 0.8v$ at the receivers TTL high state must be $\geq 2.0v$ but $\leq 5.5v$ at the receivers 4. For the iSBC 80/10 and the iSBC 80/10A use only a 1K pull-up resistor to +5v for BCLK/ and CCLK/ termination.												

**APPENDIX D**  
**BUS POWER SPECIFICATIONS**

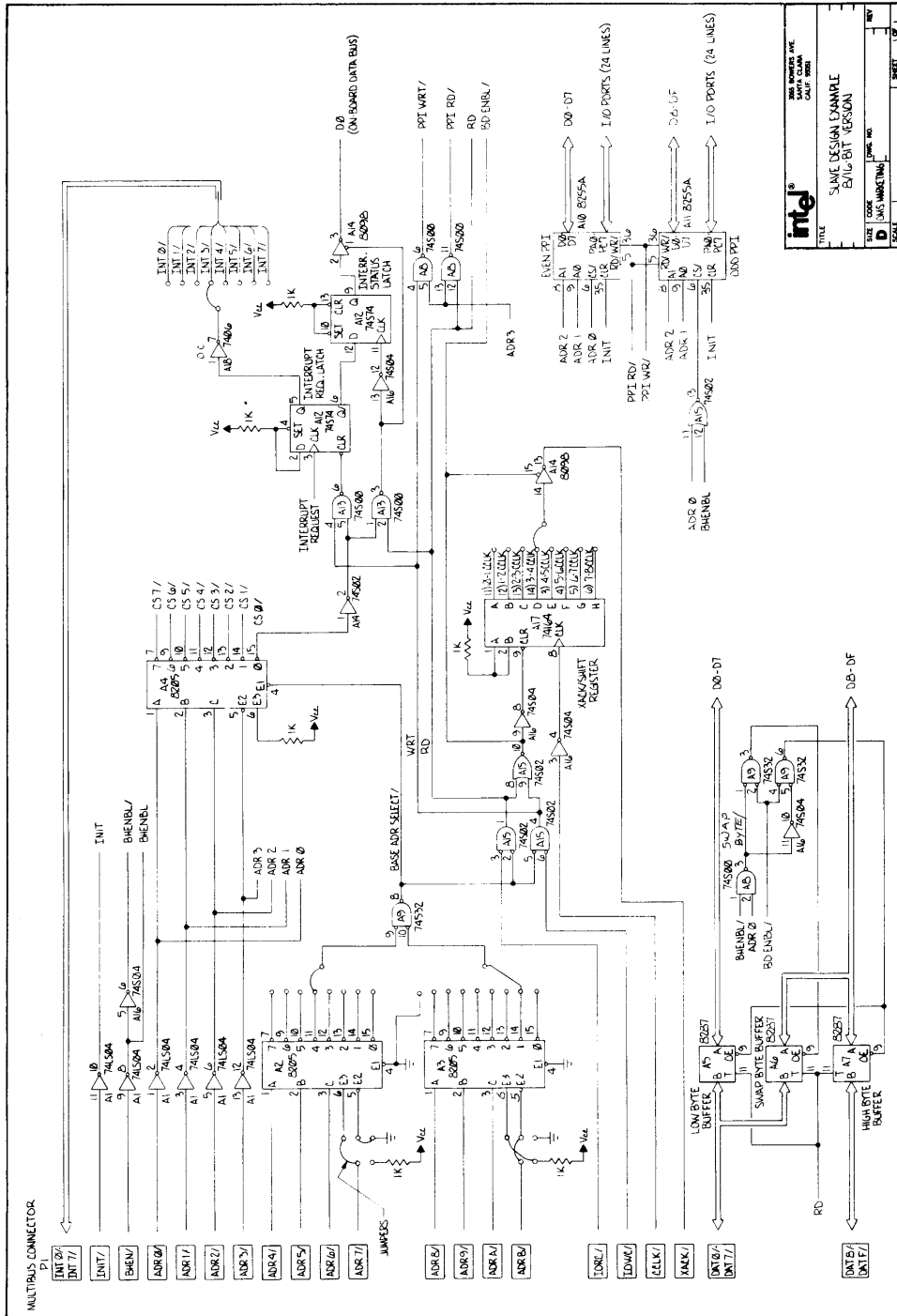
	Standard (P1)				Optional (P2)					
					Analog Power		Battery Power Backup			
	Ground	+ 5	+ 12	- 12	+ 15	- 15	+ 5	+ 12	- 12	- 5
Mnemonic	GND	+ 5V	+ 12V	- 12V	+ 15V	- 15V	+ 5B	+ 12B	- 12B	- 5B
Bus Pins	P1 + 1,2, 11,12, 75,76 85,86	P1 + 3,4, 5,6,81, 82,83, 84	P1 + 7,8	P1 + 79, 80	P2 + 23, 24	P2 + 25, 26	P2 + 3,4, 5,6	P2 + 11, 12	P2 + 15, 16	P2 - 7,8
Nominal Output	Ref.	+ 5.0V	+ 12.0V	- 12.0V	+ 15.0V	- 15.0V	+ 5.0V	+ 12.0V	- 12.0V	- 5.0V
Tolerance from Nominal <sup>1</sup>	Ref.	± 5%	± 5%	± 5%	± 3%	± 3%	± 5%	± 5%	± 5%	± 5%
Ripple (Pk-Pk) <sup>2</sup>	Ref.	50 mV	50 mV	50 mV	10 mV	10 mV	50 mV	50 mV	50 mV	50 mV
Transient Response Time <sup>3</sup>		500 $\mu$ s	500 $\mu$ s	500 $\mu$ s	100 $\mu$ s	100 $\mu$ s	500 $\mu$ s	500 $\mu$ s	500 $\mu$ s	500 $\mu$ s
Transient Deviation <sup>4</sup>		± 10%	± 10%	± 10%	± 10%	± 10%	± 10%	± 10%	± 10%	± 10%
<b>NOTES:</b> 1. Tolerance is worst case, including initial voltage setting line and load effects of power source, temperature drift, and any additional steady state influences. 2. As measured over any bandwidth not to exceed 0 to 500 kHz. 3. As measured from the start of a load change to the time an output recovers within $\pm 0.1\%$ of final voltage. 4. Measured as the peak deviation from the initial voltage.										

APPENDIX E  
MECHANICAL SPECIFICATIONS



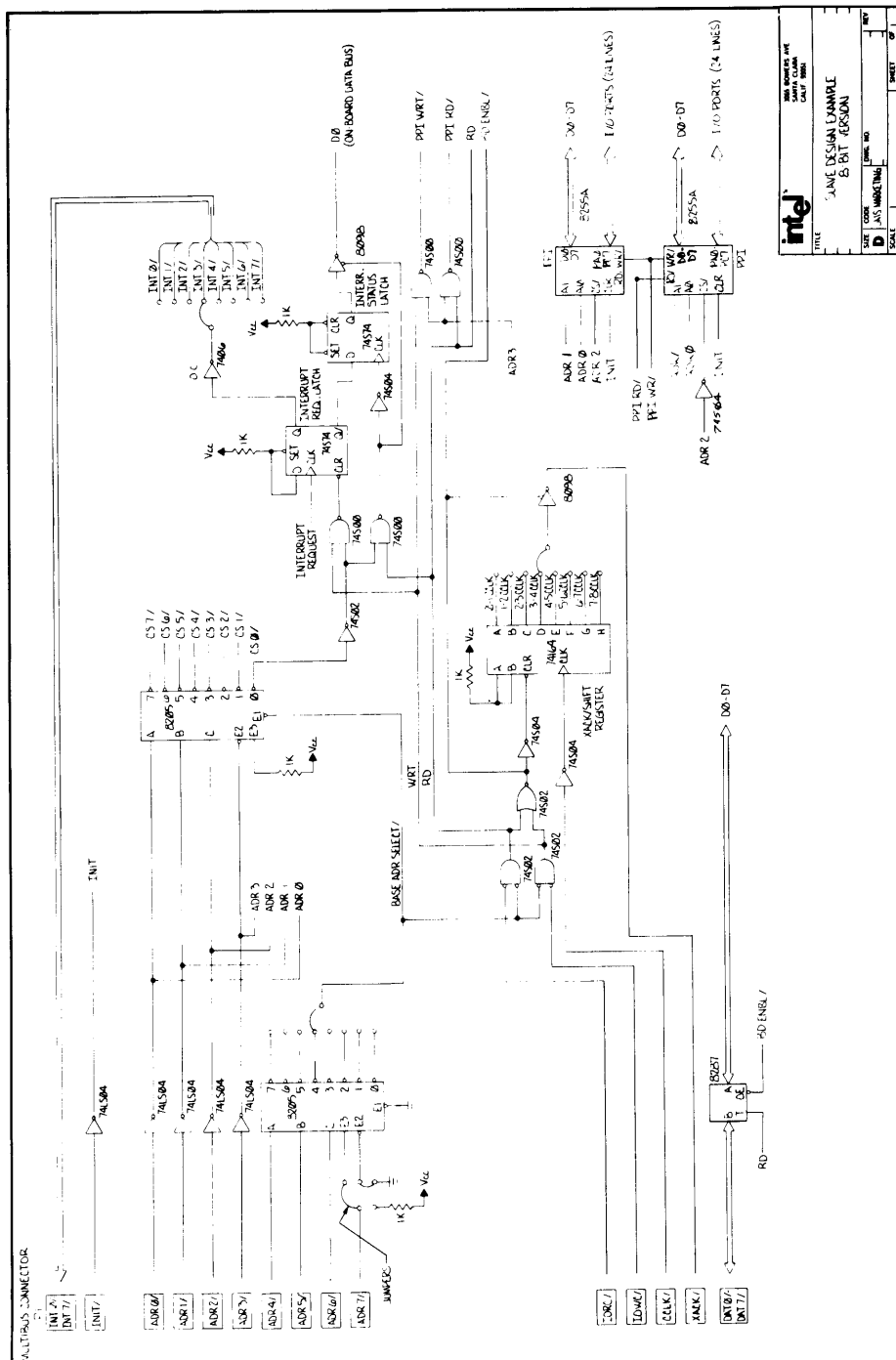
NOTES:

- 1 BOARD THICKNESS: 0.062
- 2 MULTIBUS CONNECTOR: 86-PIN, 0.156 SPACING  
CDC VF801E43D00A1  
VIKING 2VH43/1ANES
- 3 AUXILIARY CONNECTOR: 60-PIN, 0.100 SPACING  
CDC VP801B30D00A1  
TI H311130  
AMP PES-14559
- 4 EJECTOR TYPE: SCANBE #S203
- 5 BUS DRIVERS AND RECEIVERS SHOULD BE LOCATED AS CLOSE AS POSSIBLE TO THEIR RESPECTIVE MULTIBUS PIN CONNECTIONS
- 6 BOARD SPACING: 0.6
- 7 COMPONENT HEIGHT: 0.4
- 8 CLEARANCE ON CONDUCTOR NEAR EDGES: 0.050





## APPENDIX G



**MULTIBUS™ SLAVE DESIGN EXAMPLE SCHEMATIC 8-BIT VERSION**



November 1978

# Using The iSBC<sup>™</sup> 957 Execution Vehicle For Executing 8086 Program Code

Joe Barthmaier  
OEM Microcomputer Systems Applications

## 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

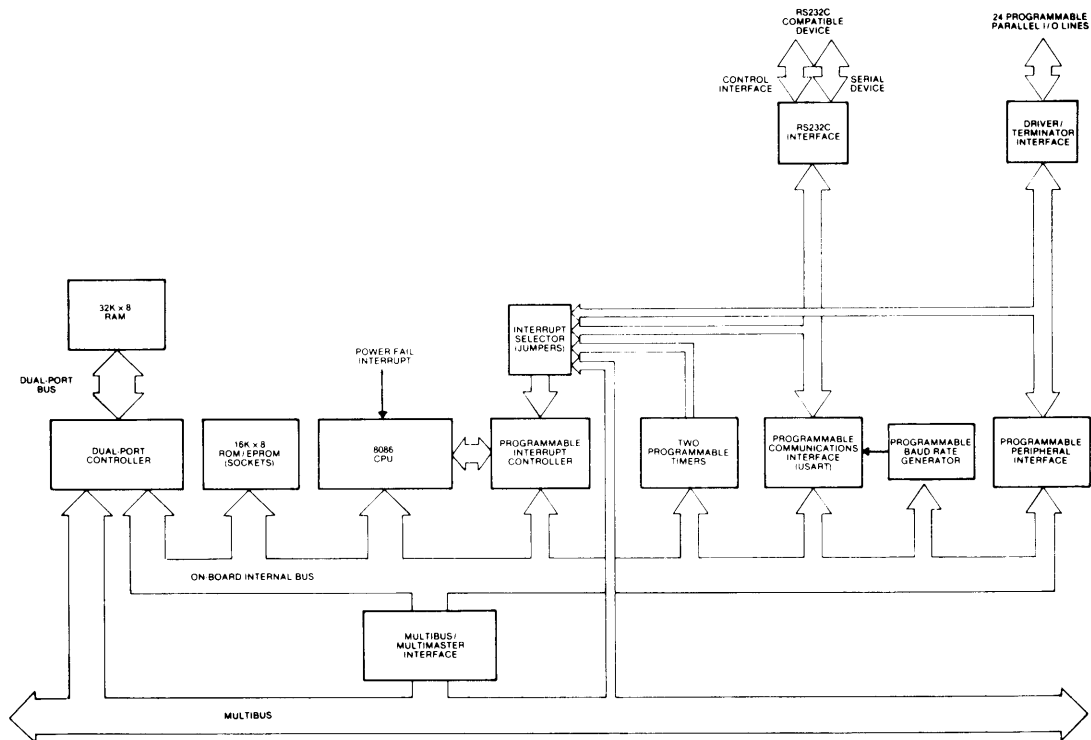


Figure 1. iSBC™ 86/12 Single Board Computer Block Diagram

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  $1.2\mu\text{sec}$  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 00000H. 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,



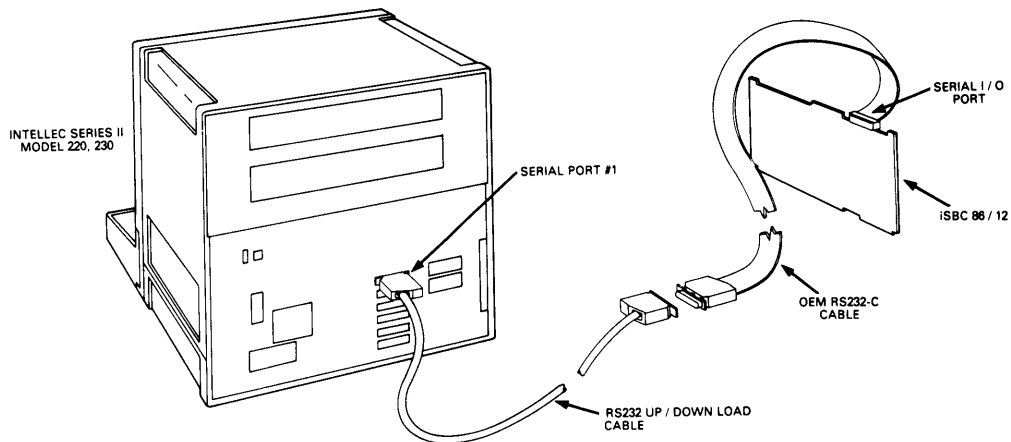


Figure 2. Intellec™ Series II Model 220, 230—iSBC™ 86/12 Configuration

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.

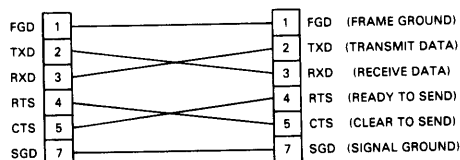


Figure 3. Intellec™—iSBC™ 86/12 RS232 UP / DOWN LOAD Cable

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 key-

board 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

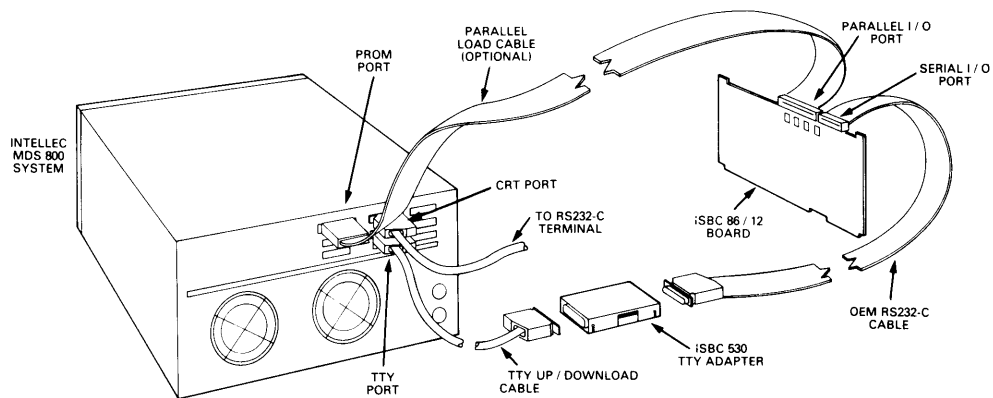
variables. The first variable is whether the iSBC 86/12 board is connected to the Intellec 800 TTY port or to the Intellec 800 CRT port. The second variable is whether or not a parallel cable is used for uploading and downloading hexadecimal object files. Figures 4A and 4B illustrate the four configurations.

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-

4A



4B

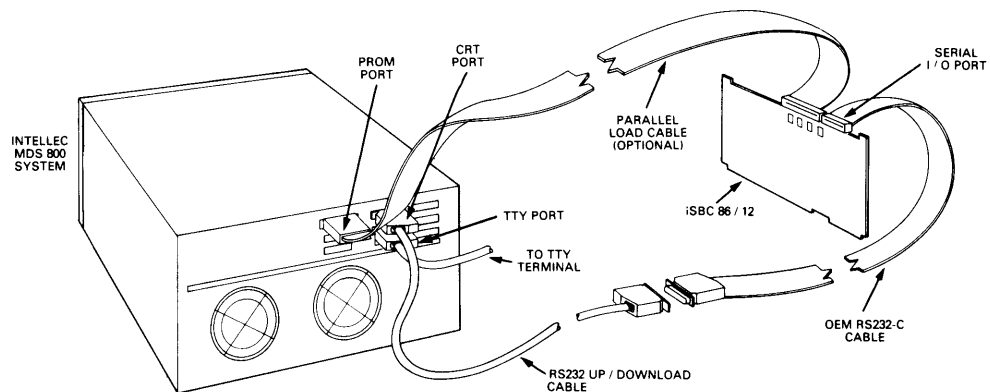


Figure 4A, 4B. Intellec™ 800—iSBC™ 86/12 Configurations

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

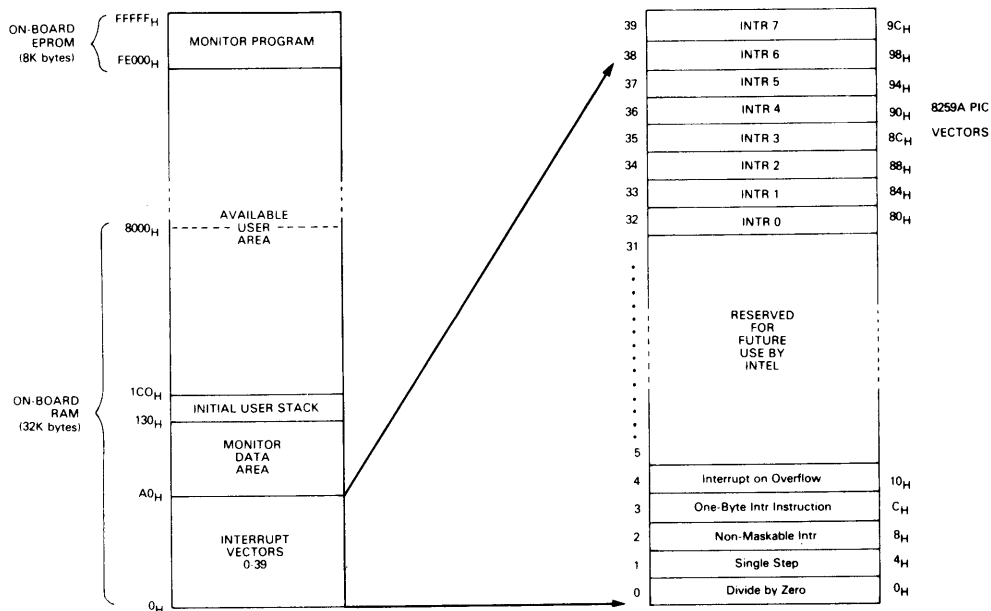


Figure 5. Memory Map of iSBC™ 86/12 Memory With Monitor Program

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 0 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 0; and the SP is set to 01C0H 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 0 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 0 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 0 connection to IR2. If the 8253 counter 0 is not specifically initialized with software, a low frequency square wave output will exist at counter 0'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., (A)) 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.

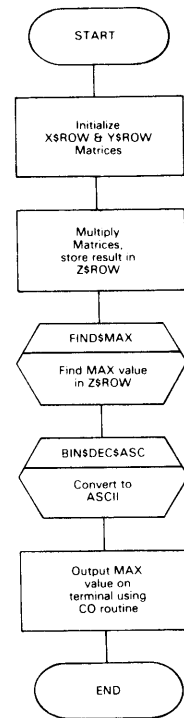


Figure 6.  
Flow Diagram of Matrix Multiplication Example

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

- (A) 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.
- (B) The next section of code contains the statements which initialize the two matrices that will be multiplied XSROW and Y\$ROW.

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

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

X\$ROW (6X5)

0	-1	-2
0	-1	-2
0	-1	-2
0	-1	-2
0	-1	-2

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

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

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.

0	0	0
0	-5	-10
0	-10	-20
0	-15	-30
0	-20	-40
0	-25	-50

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.

- (J) 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.
- (K) 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.
- (L) 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

- (M) 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 NO-LIST 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.

- (N) 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 (0100H, 0002H) indicating a CS value of 0100H and an IP value of 0002H or an absolute value of 01002H. 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.

- (P) 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=012AH, displacement = 000CH or an absolute address of 0136H. 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=0100H, IP=0B5H or 011B5H.

#### 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=0000 BX=0000 CX=0000 DX=0000 SP=01C0 BP=0000 SI=0000
DI=0000 CS=0000 DS=0000 SS=0000 ES=0000 IP=0000 FL=0000
```

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:.

```
.L, :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=0000 BX=0000 CX=0000 DX=0000 SP=01C0 BP=0000 SI=0000
DI=0000 CS=0100 DS=0000 SS=0000 ES=0000 IP=0002 FL=0000
```

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.

```
.00,100
0000 2A 01 FA 2E 8E 16 00 00 BC D0 00 8B EC 16 1F FB
0010 C7 06 8E 00 00 00 81 3E 8E 00 05 00 7E 03 E9 3C
0020 00 C7 06 90 00 00 00 31 3E 90 00 04 00 7E 03 E9
0030 22 00 8B 06 3E 00 B9 0A 00 F7 E9 8B 36 90 00 D1
0040 E6 89 C3 8B 8E 8E 00 89 88 10 00 81 06 90 00 01
0050 00 E9 D3 FF 81 06 8E 00 01 00 E9 89 FF C7 06 8E
0060 00 00 00 81 3E 8E 00 04 00 7E 03 E9 40 00 C7 06
0070 90 00 00 00 81 3E 90 00 02 00 7E 03 E9 26 00 8B
0080 06 90 00 F7 D8 50 8B 06 8E 00 89 06 00 F7 E9 8B
0090 36 90 00 D1 E6 89 C3 59 89 88 4C 00 81 06 90 00
00A0 01 00 E9 CF FF 81 06 8E 00 01 00 E9 85 FF C7 06
00B0 92 00 00 00 81 3E 92 00 02 00 7E 03 E9 8C 00 C7
00C0 06 8E 00 00 81 3E 8E 00 05 00 7E 03 E9 72 00
00D0 8B 06 8E 00 89 06 00 7F E9 8B 36 92 00 D1 E6 89
00E0 C3 C7 00 6A 00 00 00 C7 06 90 00 00 00 81 3E 90
00F0 00 04 00 7E 03 E9 41 00 8B 06 8E 00 B9 0A 00 F7
0100 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
01B4 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 0002- FA
MAX VALUE = -00050
@0100:01B5 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 012A. (012A0H is the address of the DGROUP as shown in the memory map.)

```
.X
AX=0030 BX=0005 CX=000A DX=0000 SP=00D0 BP=00D0 SI=0001
DI=0006 CS=0100 DS=012A SS=012A ES=0000 IP=01B5 FL=F202
```

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 X\$ROW, Y\$ROW, 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 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:16A,5C
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 (002) and to specify a breakpoint at address 0AEH, 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 00100:00AE 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:11A, 0001- ,
001C 0001- ,
001E 0001- ,
.SW DS:15A, FFFF- ,
003C FFFF- ,
005E 0000- ,
0060 FFFF- ,
```

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 = +00400
00100:01B5 55
.DW DS:10,4C
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 0005 0005
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 DGROUPE 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 80000H 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.

```

27 1      /* MATRIX DIMENSIONS */
28 1      DECLARE M LITERALLY '100';
29 1      DECLARE N LITERALLY '50';
        DECLARE P LITERALLY '70';

```

```

36 1      DO I = 0 TO (M-1);
37 2          DO J = 0 TO (N-1);
38 3              X$ROW(I).COL(J) = 2;
39 3          END;
40 2      END;

```

```

41 1      DO I = 0 TO (N-1);
42 2          DO J = 0 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:MATRIY.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(1000H),CONST(80000H),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 (0100H,0002H)

```

START	LTH	ALIGN	NAME	CLASS
01000H	298H	G	/GS/ CGROUP	
01000H	21DH	W	CODE (EXECUTIONVEHICLE)	CODE
0121DH	41H	B	CODE (FIND)	CODE
0125EH	3AH	W	CODE (SBCCO)	CODE
			/GE/ CGROUP	
80000H	7970H	G	/GS/ DGROUP	
80000H	CH	W	CONST (EXECUTIONVEHICLE)	CONST
8000CH	0H	W	CONST (SBCCO)	CONST
8000CH	7928H	W	DATA (EXECUTIONVEHICLE)	DATA
87936H	2H	W	DATA (FIND)	DATA
87938H	0H	W	DATA (SBCCO)	DATA
87940H	30H	SW	STACK	STACK
87970H	0H	W	MEMORY	MEMORY
			/GE/ DGROUP	
87970H	0H	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.

```

-SBC061
ISIS-II ISBC 86/12 LOADER, V1.2

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

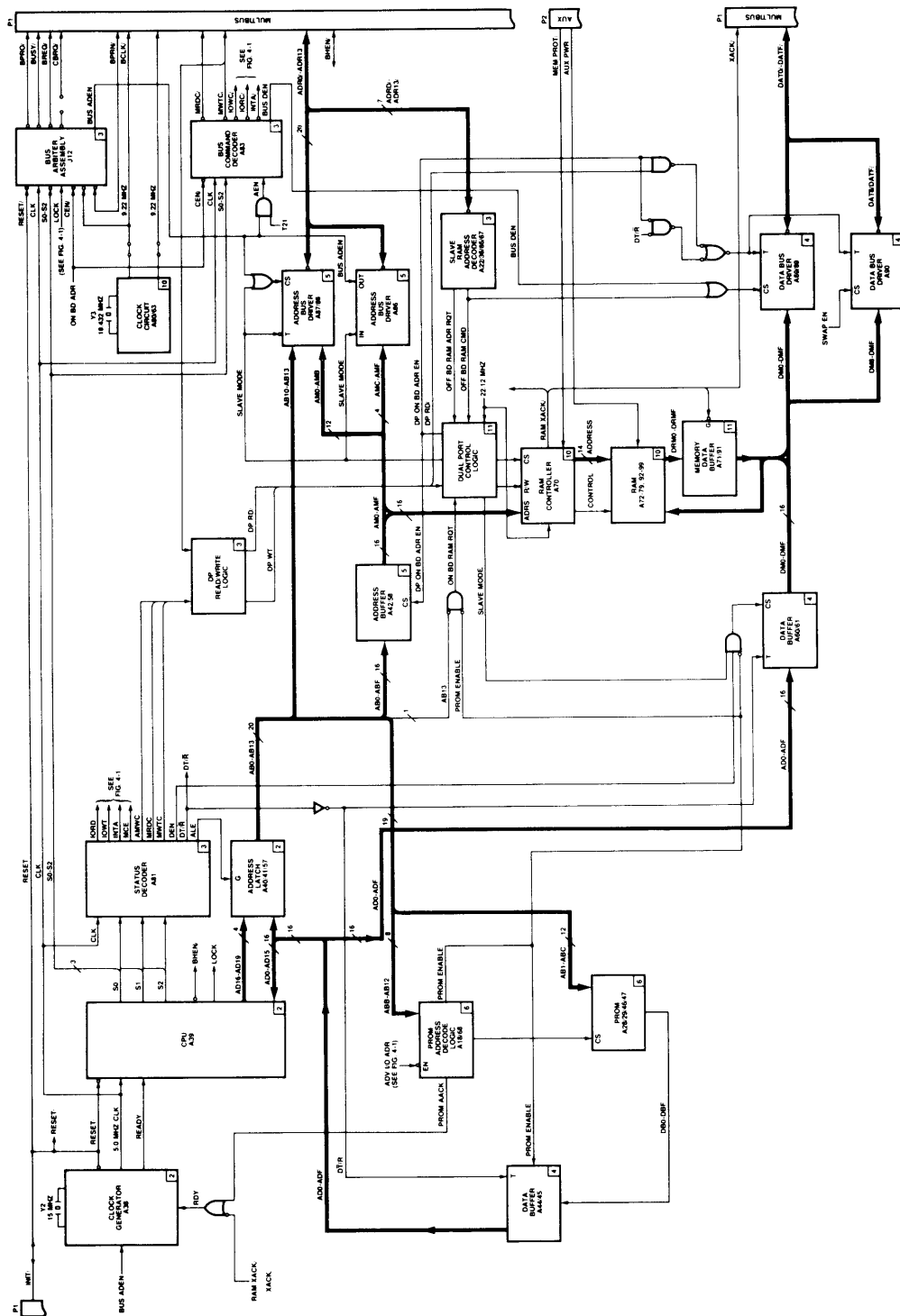
.SIAC F4- CC
.Q 0002- FA
MAX VALUE = +00300
00100:01AD 55
.

```

## VI. CONCLUSION

This application note has described the iSBC 957 Inteltec—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.





# APPENDIX B PROGRAM LISTINGS FOR EXECUTION\$VEHICLE AND FIND MODULES

PL/M-86 COMPILER EXECUTIONVEHICLE

ISIS-II PL/M-86 V1.0 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;

/*  FINDSMX - 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  FINDSMX: PROCEDURE (MATRIX$PTR, ROWS, COLS) INTEGER EXTERNAL;
3  2  DECLARE (ROWS, COLS) INTEGER;
4  2  DECLARE MATRIX$PTR POINTER;
5  2  END FINDSMX;

/*  BIN$DECSASC - BINARY TO DECIMAL ASCII CONVERSION PROCEDURE
   PARAMETERS:
   VALUE - INTEGER VALUE TO BE CONVERTED TO ASCII
   CHARSARRAY$ADR - ADDRESS OF 6 BYTE ARRAY WHERE ASCII
   STRING CONTAINING THE VALUE WILL BE STORED

*/
6  1  BIN$DECSASC: PROCEDURE (VALUE, CHARSARRAY$ADR);

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

10 2  IF VALUE < 0 THEN
11 2  DO;
12 3  CHARSARRAY(0) = '-'; /* SIGN CHARACTER */
13 3  TEMP = -VALUE;
14 3  END;
15 2  DO;
16 3  CHARSARRAY(0) = '+';
17 3  TEMP = VALUE;
18 3  END;
19 2  DO I = 5 TO 1 BY -1;
20 3  CHARSARRAY(I) = UNSIGN(TEMP MOD 10) + 30H;
21 3  TEMP = TEMP/10;
/* ASCII CHARACTERS 20 THRU 29 HEX REPRESENT THE DIGITS 0 THRU 9. THUS
   TO CONVERT AN INTEGER TO ASCII REPEATED DIVISIONS BY 10 AND ADDING
   THE REMAINDER TO 30 HEX WILL ACCOMPLISH THE CONVERSION */
22 3  END;
23 2  END BIN$DECSASC;

/*  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 '5';
28 1  DECLARE N LITERALLY '5';
29 1  DECLARE P LITERALLY '3';

/* THE THREE MATRICES ARE DECLARED AS ARRAYS OF STRUCTURES. XSROW IS COMPOSED
   OF M STRUCTURES EACH OF WHICH IS COMPOSED OF N INTEGER ELEMENTS. THUS
   XSROW 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$ROW IS DECLARED AS A M X P MATRIX AND Z$ROW AS A N X P MATRIX */
30 1  DECLARE XSROW(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$ASCARRAY(6) BYTE;
35 1  DECLARE TEXT(*) BYTE DATA ('MAX VALUE = ');

```



```

/* INITIALIZE XSROW SUCH THAT THE FIRST ROW IS SET EQUAL TO 0, THE SECOND
ROW EQUAL TO 1, THE THIRD ROW EQUAL TO 2, ETC. */
36 1 DO I = 0 TO (M-1);
37 2 DO J = 0 TO (N-1);
38 3 XSROW(I).COL(J) = I;
39 3 END;
40 2 END;

/* INITIALIZE YSROW SUCH THAT THE FIRST COLUMN IS SET EQUAL TO 0, THE
SECOND COLUMN EQUAL TO -1, AND THE THIRD COLUMN EQUAL TO -2. */
41 1 DO I = 0 TO (N-1);
42 2 DO J = 0 TO (P-1);
43 3 YSROW(I).COL(J) = -J;
44 3 END;
45 2 END;

/* PERFORM MATRIX MULTIPLICATION */
46 1 DO K = 0 TO (P-1);
47 2 DO I = 0 TO (M-1);
48 3 ZSROW(I).COL(K) = 0; /* SET ZSROW ELEMENT TO 0 */
49 3 DO J = 0 TO (N-1); /* SUM THE PRODUCT OF XSROW ROW TERMS AND YSROW COLUMN TERMS */
50 4 ZSROW(I).COL(K) = ZSROW(I).COL(K) + (XSROW(I).COL(J) * YSROW(J).COL(K));
51 4 END;
52 3 END;
53 2 END;

54 1 MAX = FINDSMX (@ZSROW, M, P); /* FIND MAX VALUE OF ZSROW */

55 1 CALL BINSDECSASC (MAX, @MAXSASCARRAY); /* CONVERT TO DECIMAL ASCII */

56 1 DO I = 0 TO (SIGNSD(SIZE(TEXT)) - 1); /* OUTPUT HEADER TEXT */
57 2 CALL CO(TEXT(I));
58 2 END;

59 1 DO I = 0 TO 5; /* OUTPUT ASCII MAX VALUE */
60 2 CALL CO(MAXSASCARRAY(I));
61 2 END;

62 1 END EXECUTION$VEHICLE;

```

## MODULE INFORMATION:

```

CODE AREA SIZE      = 0225H      549D
CONSTANT AREA SIZE  = 000CH      12D
VARIABLE AREA SIZE  = 0090H     144D
MAXIMUM STACK SIZE  = 0000H       0D
137 LINES READ
0 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.
10			
11			PL/M CALLING SEQUENCE:
12			MAX\$VALUE = FINDSMX(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
20			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			
30			THE ALGORITHM USED IS SIMILAR TO THE FOLLOWING PL/M CODE:
31			FOR I = 0 TO (\$OF\$ROWS - 1);
32			FOR J = 0 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			

## APPENDIX B

## PROGRAM LISTINGS FOR EXECUTION\$VEHICLE AND FIND MODULES

```

LOC  OBJ          LINE  SOURCE
-----
      40 ;
      41 ;
      42 ;   DEFINE GROUPS TO CONFORM WITH PL/M-86 CONVENTIONS. DATA, STACK, AND
      43 ;   CODE SEGMENTS WILL BE APPENDED TO THEIR RESPECTIVE SEGMENTS IN THE
      44 ;   PL/M-86 MODULES.
      45 DGROUP  GROUP  DATA,STACK
      46 CGROUP  GROUP  CODE
      47 ;
      48 ;   INSTRUCT THE ASSEMBLER THAT THE DS, SS, AND CS REGISTERS WILL CONTAIN
      49 ;   THE BASE ADDRESS VALUES FOR THE DGROUP, DGROUP AND CGROUP GROUPS.
      50 ;   ASSUME  DS:DGROUP,SS:DGROUP,CS:CGROUP
      51 ;
      52 ;
      53 ;*****DATA SEGMENT
      54 ;
      55 DATA    SEGMENT WORD PUBLIC 'DATA'
      56 MAX      DW      0
      57 DATA    ENDS
      58 ;
      59 ;*****STACK SEGMENT
      60 ;
      61 STACK   SEGMENT STACK 'STACK'
      62 DW      14 DUP (0)      ;RESERVE 13 WORDS OF STACK FOR MONITOR
      63 ;
      64 ;*****CODE SEGMENT
      65 ;
      66 CODE     SEGMENT BYTE PUBLIC 'CODE'
      67 ;
      68 ;
      69 ;
      70 ;PARAMETERS ON STACK, DISPLACEMENT FROM TOS INCREASED BY TWO DUE TO INITIAL PUSH
      71 NO_OF_ROWS EQU    WORD PTR [BP+6]
      72 NO_OF_COLS EQU    WORD PTR [BP+4]
      73 ADR_OF_MATRIX EQU  WORD PTR [BP+8]
      74 ;
      75 FINDMX PROC NEAR      ;PROCEDURE DECLARATION
      76 BP          PUSH      ;SAVE BP REGISTER
      77 MOV         BP,SP      ;BP POINTS TO PARAMETERS ON STACK
      78 XOR         DX,DX      ;SET DX = ABS OF CURRENT MAX = 0
      79 MOV         DI,DX     ;DI = I(ROW INDEX) = 0
      80 MOV         SI,DX     ;SI = J(COLUMN INDEX) = 0
      81 MOV         MAX,DX   ;MAX = CURRENT MAX = 0
      82 MOV         CX,NO_OF_COLS
      83 SHL         CX,1     ;CX = (#OFSCOLS) * 2
      84 ;
      85 MOV         BX,ADR_OF_MATRIX ;TERMINATION FOR J(SI) INDEX
      86 ;
      87 ABC:      MOV         AX,[BX][SI] ;GET ELEMENT OF MATRIX
      88 OR          AX,AX      ;SET FLAGS
      89 JNS         DEF       ;JUMP IF SIGN = 0
      90 NEG         AX        ;NEGATE TO FORM POSITIVE NUMBER
      91 DEF:      CMP         AX,DX   ;COMPARE TO CURRENT MAX
      92 JL         XYZ       ;JUMP IF LESS THAN CURRENT MAX
      93 MOV         DX,AX      ;MOVE TO ABS OF CURRENT MAX
      94 MOV         AX,[BX][SI] ;MOVE MATRIX VALUE TO CURRENT MAX
      95 MOV         MAX,AX
      96 XYZ:      ADD         SI,2   ;INCREMENT J INDEX BY TWO
      97 CMP         SI,CX      ;END OF THIS ROW ??
      98 JB         ABC        ;IF NO, LOOP BACK FOR NEXT ELEMENT OF THIS ROW
      99 LEA         BX,[BX+SI] ;BX = BX + (2 * #OFSCOLS), BX POINTS TO NEXT ROW
      100 MOV         SI,0      ;J = 0
      101 INC         DI        ;I = I + 1
      102 CMP         DI,NO_OF_ROWS ;LAST ROW ??
      103 JB         ABC        ;IF NO, DO THE NEXT ROW
      104 MOV         AX,MAX     ;RETURN MAX VALUE IN AX REGISTER
      105 POP         BP        ;RESTORE BP REGISTER
      106 RET         6        ;INCREMENT SP BY 6 AND RETURN TO CALLER
      107 ;
      108 FINDMX ENDP
      109 ;
      110 CODE     ENDS
      111 ;
      112 END

```

## SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
??SEG . . . .	SEGMENT		SIZE=0000H PARA PUBLIC
ABC . . . . .	L NEAR	0015H	CODE
ADR_OF_MATRIX	V WORD	0008H	[BP]
DGROUP . . . .	GROUP		
CODE . . . . .	SEGMENT		SIZE=0041H BYTE PUBLIC 'CODE'
DATA . . . . .	SEGMENT		SIZE=0002H WORD PUBLIC 'DATA'
DEF . . . . .	L NEAR	001DH	CODE
DGROUP . . . .	GROUP		DATA STACK
FINDMX . . . .	L NEAR	0000H	CODE PUBLIC
MAX . . . . .	V WORD	0000H	DATA
NO_OF_COLS . .	V WORD	0004H	[BP]
NO_OF_ROWS . .	V WORD	0006H	[BP]
STACK . . . . .	SEGMENT		SIZE=001CH PARA STACK 'STACK'
XYZ . . . . .	L NEAR	0020H	CODE

ASSEMBLY COMPLETE, NO ERRORS FOUND

ISIS-II QRL-P6, V3.1

INVOKED BY:

QRLR6 : F1: MATRIX.OBJ, : F1: FIND.OBJ, SBCIOS.LIB ORIGIN(1000H)

INPUT MODULES INCLUDED:

: F1: MATRIX.OBJ (EXECUTIONVEHICLE)

: F1: FIND.OBJ (FIND)

SBCIOS.LIB (SBCCO)

RESULT WRITTEN TO : F1: MATRIX (EXECUTIONVEHICLE)

START ADDRESS IS (0100H, 0002H)

(N)

START	LTH	ALIGN	NAME	CLASS
01000H	20H	G	/GS/ CGROUP	
01000H	225H	W	CODE (EXECUTIONVEHICLE)	CODE
010225H	41H	B	CODE (FIND)	CODE
01266H	3AH	W	CODE (SBCCO)	CODE
			/GE/ CGROUP	
012A0H	D0H	G	/GS/ DGROUP	
012A0H	CH	W	CONST (EXECUTIONVEHICLE)	CONST
012ACH	0H	W	CONST (SBCCO)	CONST
012ACH	90H	W	DATA (EXECUTIONVEHICLE)	DATA
0133CH	2H	W	DATA (FIND)	DATA
0133EH	0H	W	DATA (SBCCO)	DATA
01340H	70H	SW	STACK	STACK
01370H	0H	W	MEMORY	MEMORY
			/GE/ DGROUP	
01370H	0H	G	??SEG (FIND)	(NULL)

DEBUG MAP OF : F1: MATRIX (EXECUTIONVEHICLE)

(P)

MODULE: EXECUTIONVEHICLE	0100H, 01E1H	LINE #: 19	0100H, 0139H	LINE #: 52	
012AH, 00D0H	SYMBOL: MEMORY	0100H, 01FBH	LINE #: 20	0100H, 0142H	LINE #: 53
0100H, 01B5H	SYMBOL: BINDECASC	0100H, 0213H	LINE #: 21	0100H, 014BH	LINE #: 54
012AH, 000CH	SYMBOL: TEMP	0100H, 021EH	LINE #: 22	0100H, 015EH	LINE #: 55
012AH, 000EH	SYMBOL: I	0100H, 0221H	LINE #: 23	0100H, 0169H	LINE #: 56
012AH, 0010H	SYMBOL: XROW	0100H, 0002H	LINE #: 36	0100H, 017AH	LINE #: 57
012AH, 004CH	SYMBOL: YROW	0100H, 0021H	LINE #: 37	0100H, 0185H	LINE #: 58
012AH, 006AH	SYMBOL: ZROW	0100H, 0032H	LINE #: 38	0100H, 018EH	LINE #: 59
012AH, 008EH	SYMBOL: I	0100H, 004BH	LINE #: 39	0100H, 019FH	LINE #: 60
012AH, 0090H	SYMBOL: J	0100H, 0054H	LINE #: 40	0100H, 01AAH	LINE #: 61
012AH, 0092H	SYMBOL: K	0100H, 005DH	LINE #: 41	0100H, 01B3H	LINE #: 62
012AH, 0094H	SYMBOL: MAX	0100H, 006EH	LINE #: 42		
012AH, 0096H	SYMBOL: MAXASCARRAY	0100H, 007FH	LINE #: 43	0100H, 023AH	SYMBOL: FIN
012AH, 0000H	SYMBOL: TEXT	0100H, 009CH	LINE #: 44	0100H, 0242H	SYMBOL: ABC
0100H, 01B5H	LINE #: 6	0100H, 00A5H	LINE #: 45	0100H, 0225H	SYMBOL: DEF
0100H, 01B8H	LINE #: 10	0100H, 00AEH	LINE #: 46	012AH, 009CH	SYMBOL: FINDMX
0100H, 01C2H	LINE #: 12	0100H, 00BFH	LINE #: 47	0100H, 009CH	SYMBOL: MAX
0100H, 01C8H	LINE #: 13	0100H, 00D0H	LINE #: 48	0100H, 024DH	SYMBOL: XYZ
0100H, 01D1H	LINE #: 14	0100H, 00E7H	LINE #: 49	0100H, 0225H	PUBLIC: FINDMX
0100H, 01DAH	LINE #: 16	0100H, 00F8H	LINE #: 50		MODULE: SBCCO
0100H, 01DAH	LINE #: 17	0100H, 0130H	LINE #: 51	0100H, 0266H	PUBLIC: CO

## 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$MX - EXTERNAL ASSEMBLY LANGUAGE PROCEDURE WHICH SEARCHES A
   MATRIX FOR THE LARGEST ABSOLUTE MAGNITUDE.
   PARAMETERS:
   MATRIX$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;

/* BINSDECSASC - BINARY TO DECIMAL ASCII CONVERSION PROCEDURE
   PARAMETERS:
   VALUE - INTEGER VALUE TO BE CONVERTED TO ASCII
   CHARS$ARRAY$ADR - ADDRESS OF 6 BYTE ARRAY WHERE ASCII
   STRING CONTAINING THE VALUE WILL BE STORED
*/
6  1  BINSDECSASC: PROCEDURE (VALUE, CHARS$ARRAY$ADR);
   ; STATEMENT # 5
               BINDECSASC      PROC NEAR
01B5  55      PUSH      BP
01B6  8BEC    MOV       BP,SP

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

10 2  IF VALUE < 0 THEN
               ; STATEMENT # 10
01B8  817E060000 CMP     [BP].VALUE,0H
01BD  7C03     JL      S+5H
01BF  E91200   JMP     @1
11 2  DO;
12 3  CHARS$ARRAY(0) = '-'; /* SIGN CHARACTER */
               ; STATEMENT # 12
01C2  8B5E04   MOV     BX,[BP].CHARS$ARRAY$ADR
01C5  C6077D   MOV     CHARARRAY[BX],2DH
13 3  TEMP = -VALUE;
               ; STATEMENT # 13
01C8  8B4506   MOV     AX,[BP].VALUE
01CB  F7D8     NEG     AX
01CD  89F60000 MOV     TEMP,AX
14 3  END;
               ; STATEMENT # 14
01D1  E90D00   JMP     @2
               @1:
      ELSE
15 2  DO;
16 3  CHARS$ARRAY(0) = '+';
               ; STATEMENT # 16
01D4  8B5E04   MOV     BX,[BP].CHARS$ARRAY$ADR
01D7  C6077B   MOV     CHARARRAY[BX],7BH
17 3  TEMP = VALUE;
               ; STATEMENT # 17
01DA  8B4506   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  E906F0     JMP    @5
               @3:
01EA  81060200FFFF ADD     I,0FFFFH

```

```

                                #5:
P1F0 813E0700P100    CMP    I,1H
01F6 7D03             JGE    S+5H
01F8 E926P0          JMP     #4
20 3    CHARSARRAY(I) = UNSIGN(TEMP MOD 10) + 30H;
                                ; STATEMENT # 20
01FB 8B060000        MOV     AX,TEMP
01FF 890A00          MOV     CX,0AH
0202 31D2             XOR     DX,DX
0204 F7F9             IDIV    CX
0206 81C73000        ADD     DX,30H
P20A 8B5E04          MOV     BX,[BP].CHARARRAYADR
020D 8B360200        MOV     SI,I
P211 881F             MOV     [BX].CHARARRAY[SI],DL
21 3    TEMP = TEMP/10;
                                ; STATEMENT # 21
/* ASCII CHARACTERS 30 THRU 39 HEX REPRESENT THE DIGITS 0 THRU 9. THUS
TO CONVERT AN INTEGER TO ASCII REPEATED DIVISIONS BY 10 AND ADDING
THE REMAINDER TO 30 HEX WILL ACCOMPLISH THE CONVERSION */
P213 8B060000        MOV     AX,TEMP
0217 99              CWD
0218 F7F9             IDIV    CX
021A 89060000        MOV     TEMP,AX
22 3    END;
                                ; STATEMENT # 22
021E E9C9FF          JMP     #3
                                #4:
23 2    END BINDEC$ASC;
                                ; STATEMENT # 23
0221 5D              POP     BP
0222 C20400          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
*/
CO: PROCEDURE (CHAR) EXTERNAL;
24 1    DECLARE CHAR BYTE;
25 2    END CO;
26 2

/* 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.  XSROW IS COMPOSED
OF M STRUCTURES EACH OF WHICH IS COMPOSED OF N INTEGER ELEMENTS.  THUS
XSROW 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.  YSROW IS DECLARED AS A N X P MATRIX AND ZSROW AS A N X P MATRIX */
30 1    DECLARE XSROW(M) STRUCTURE (COL(N) INTEGER);
31 1    DECLARE YSROW(N) STRUCTURE (COL(P) INTEGER);
32 1    DECLARE ZSROW(M) STRUCTURE (COL(P) INTEGER);

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

/* INITIALIZE XSROW SUCH THAT THE FIRST ROW IS SET EQUAL TO 0, THE SECOND
ROW EQUAL TO 1, THE THIRD ROW EQUAL TO 2, ETC. */
36 1    DO I = 0 TO (M-1);
                                STATEMENT # 36
0002 FA              CLI
0003 2E0F160000      MOV     SS,CS:@@STACK$FRAME
0008 BCD000          MOV     SP,@@STACK$OFFSET
000B 8BEC            MOV     BP,SP
000D 16              PUSH    SS
000E 1F              POP     DS
000F FB              STI
0010 C70602000000    MOV     I,PH
                                #6:
0016 813E0200P500    CMP     I,5H
001C 7EP3            JLE     S+5H
001E E93CP0          JMP     #7
37 2    DO J = 0 TO (N-1);
                                ; STATEMENT # 37
0021 C70604000000    MOV     J,0H
                                #7:
0027 813E04000040P0  CMP     J,4H
002D 7EP3            JLE     S+5H
002F E9220F          JMP     #9
38 3    XSROW(I).COL(J) = I;
                                ; STATEMENT # 38
0032 8B060200        MOV     AX,I
0036 B9FA00          MOV     CX,0AH
0039 F7E9            IMUL    CX
003B 8B360400        MOV     SI,J
003F D1E6            SHL     SI,1
0041 89C3            MOV     BX,AX
0043 8B0E0200        MOV     CX,I
0047 89080400        MOV     [BX].XSROW[SI],CX
39 3    END;

```

```

                                ; STATEMENT # 39
004B 210684000100 ADD J,1H
0051 E9D3FF JMP #8

40 2 END;                                ; STATEMENT # 40
0054 210682000100 ADD I,1H
005A E9B9FF JMP #6
007:

/* INITIALIZE YSROW SUCH THAT THE FIRST COLUMN IS SET EQUAL TO 0, THE
SECOND COLUMN EQUAL TO -1, AND THE THIRD COLUMN EQUAL TO -2. */
41 1 DO I = 0 TO (N-1);                                ; STATEMENT # 41
005D C70682000020 MOV I,0H
010:
0063 813E82000400 CMP I,4H
0069 7E03 JLE S+5H
006B E94000 JMP #11
42 2 DO J = 0 TO (P-1);                                ; STATEMENT # 42
006E C70684000000 MOV J,0H
012:
0074 813E84000200 CMP J,2H
007A 7E03 JLE S+5H
007C E92600 JMP #13
43 3 YSROW(I).COL(J) = -J;                                ; STATEMENT # 43
007F 8B068400 MOV AX,J
0083 F7D8 NEG AX
0085 50 PUSH AX ; 1
0086 8B068200 MOV AX,I
008A B90600 MOV CX,6H
008D F7E9 IMUL CX
008F 8B368400 MOV SI,J
0093 D1E6 SHL SI,1
0095 89C3 MOV BX,AX
0097 59 POP CX ; 1
0098 898B4000 MOV [BX].YROW(SI),CX
44 3 END;                                ; STATEMENT # 44
009C 810684000100 ADD J,1H
00A2 E9CFFF JMP #12
013:
45 2 END;                                ; STATEMENT # 45
00A5 210682000100 ADD I,1H
00AB E9B5FF JMP #10
011:

/* PERFORM MATRIX MULTIPLICATION */
46 1 DO K = 0 TO (P-1);                                ; STATEMENT # 46
00AE C70686000000 MOV K,0H
014:
00B4 813E86000200 CMP K,2H
00BA 7E03 JLE S+5H
00BC E92C00 JMP #15
47 2 DO I = 0 TO (M-1);                                ; STATEMENT # 47
00BF C70682000000 MOV I,0H
016:
00C5 813E82000500 CMP I,5H
00CB 7E03 JLE S+5H
00CD E97200 JMP #17
48 3 ZSROW(I).COL(K) = 0; /* SET ZSROW ELEMENT TO 0 */ ; STATEMENT # 48
00D0 8B068200 MOV AX,I
00D4 B90600 MOV CX,6H
00D7 F7E9 IMUL CX
00D9 8B368600 MOV SI,K
00DD D1E6 SHL SI,1
00DF 89C3 MOV BX,AX
00E1 C7065E000000 MOV [BX].ZROW(SI),0H
49 3 DO J = 0 TO (N-1); /* SUM THE PRODUCT OF XSROW ROW TERMS AND YSROW COLUMN TERMS */ ; STATEMENT # 49
00E7 C70684000000 MOV J,0H
018:
00ED 813E84000400 CMP J,4H
00F3 7E03 JLE S+5H
00F5 E94100 JMP #19
50 4 ZSROW(I).COL(K) = ZSROW(I).COL(K) + ( XSROW(I).COL(J) * YSROW(J).COL(K) ); ; STATEMENT # 50
00F8 8B068200 MOV AX,I
00FC 890A00 MOV CX,0AH
00FF F7E9 IMUL CX
0101 8B368400 MOV SI,J
0105 D1E6 SHL SI,1
0107 50 PUSH AX ; 1
0108 8B068400 MOV AX,J
010C B90600 MOV CX,6H
010F F7E9 IMUL CX
0111 8B3E8600 MOV DI,K
0115 D1E7 SHL DI,1
0117 89C3 MOV BX,AX
0119 8B814000 MOV AX,[BX].YROW(DI)
011D 5B POP BX ; 1
011E F7A80400 IMUL [BX].XROW(SI)
0122 50 PUSH AX ; 1
0123 8B068200 MOV AX,I
0127 F7E9 IMUL CX
0129 89C3 MOV BX,AX

```

```

      @12B 58      POP      AX      ; 1
      @12C @1815E00 ADD      [BX],ZROW[DI],AX
51  4      END;
      ; STATEMENT # 51
      @130 810684000100 ADD      J,1H
      @135 E9B4FF      JMP      @18
      @19:
52  3      END;
      ; STATEMENT # 52
      @139 810684000100 ADD      I,1H
      @13F E983FF      JMP      @16
      @17:
53  2      END;
      ; STATEMENT # 53
      @142 810686000100 ADD      K,1H
      @148 E969FF      JMP      @14
      @15:
54  1      MAX = FINDSMX (@ZCROW, M, P); /* FIND MAX VALUE OF Z$ROW */
      ; STATEMENT # 54
      @14B B85E00      MOV      AX,OFFSET(ZROW)
      @14E 50      PUSH     AX      ; 1
      @14F B80600      MOV      AX,6H
      @152 50      PUSH     AX      ; 2
      @153 B80300      MOV      AX,3H
      @156 50      PUSH     AX      ; 3
      @157 E82000      CALL    FINDMX
      @15A 80068000      MOV      MAX,AX
55  1      CALL BINSDECASC (MAX, @MAX$ASC$ARRAY); /* CONVERT TO DECIMAL ASCII */
      ; STATEMENT # 55
      @15E FF368000      PUSH     MAX      ; 1
      @162 B88A00      MOV      AX,OFFSET(MAX$ASC$ARRAY)
      @165 50      PUSH     AX      ; 2
      @166 E84C00      CALL    BINDECASC
56  1      DO I = 0 TO (SIGNED(SIZE(TEXT)) - 1); /* OUTPUT HEADER TEXT */
      ; STATEMENT # 56
      @169 C70682000000 MOV      I,0H
      @20:
      @16F @13E82000000 CMP      I,0BH
      @175 7E03      JLE      $+5H
      @177 E91400      JMP      @21
57  2      CALL CO(TEXT(I));
      ; STATEMENT # 57
      @17A 8B1E8200      MOV      BX,I
      @17E FFB70000      PUSH     TEXT[BX]; 1
      @182 E80000      CALL    CO
58  2      END;
      ; STATEMENT # 58
      @185 810682000100 ADD      I,1H
      @188 E9E1FF      JMP      @20
      @21:
59  1      DO I = 0 TO 5; /* OUTPUT ASCII MAX VALUE */
      ; STATEMENT # 59
      @18E C70682000000 MOV      I,0H
      @22:
      @194 813E820000500 CMP      I,5H
      @19A 7EP3      JLE      $+5H
      @19C E91400      JMP      @23
60  2      CALL CO(MAX$ASC$ARRAY(I));
      ; STATEMENT # 60
      @19F 8B1E8200      MOV      BX,I
      @1A3 FFB78A00      PUSH     MAX$ASC$ARRAY[BX]; 1
      @1A7 E80000      CALL    CO
61  2      END;
      ; STATEMENT # 61
      @1AA 810682000100 ADD      I,1H
      @1B0 E9E1FF      JMP      @22
      @23:
62  1      END EXECUTION$VEHICLE;
      ; STATEMENT # 62
      @1B3 FB      STI
      @1B4 F4      HLT

```

## MODULE INFORMATION:

```

CODE AREA SIZE      = 0225H      549D
CONSTANT AREA SIZE  = 000CH      12D
VARIABLE AREA SIZE  = 0090H      144D
MAXIMUM STACK SIZE  = 0000H      8D
137 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-86 COMPILATION

