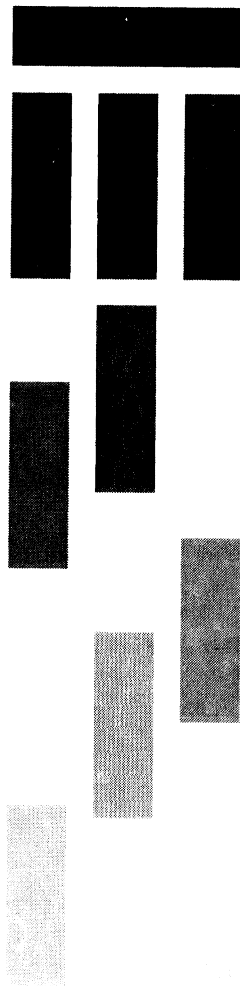
# *EV80C196KB Evaluation Board User's Manual*



intel®

*Order Number 270738-001*

# EV80C196KB Microcontroller Evaluation Board

## USER'S MANUAL

**Release 001**

**February 20, 1989**

Copyright 1989, Intel Corporation

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibitlity for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patents licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication, or disclosure is subject to restrictions stated in Intel's Software License Agreement, or in the case of software delivered to the U.S. government, in accordance with the software license agreement as defined in FAR 52.227-7013.

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel Corporation.

Intel Corporation retains the right to make changes to these specifications at any time, without notice. Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

376, Above, ActionMedia, BITBUS, Code Builder, DeskWare, Digital Studio, DVI, EtherExpress, ETOX, ExCA, FaxBACK, Grand Challenge, i, i287, i386, i387, i486, i487, i750, i860, i960, ICE, iLBX, Inboard, Intel, Intel287, Intel386, Intel387, Intel486, Intel487, intel inside., Intellec, iPSC, iRMX, iSBC, iSBX, iWarp, LANprint, LANSelect, LANShell, LANSight, LANSpace, LANSpool, MAPNET, Matched, MCS, Media Mail, NetPort, NetSentry, OpenNET, Paragon, PRO750, ProSolver, READY-LAN, Reference Point, RMX/80, SatisFAXtion, SnapIn 386, Storage Broker, SugarCube, The Computer Inside., TokenExpress, Visual Edge, and WYPIWYF.

and the combinations of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corportation.

CHMOS and HMOS are patented processes of Intel Corp.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

IBM is a registered trademark and AT is a trademark of International Business Machines, Inc.

Microsoft, MS, MS-DOS, and XENIX are registered trademarks and Multiplan is a trademark of Microsoft Corporation.

Addional copies of this manual or other Intel Literature may be obtained from:

Intel Corporation, Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

# CONTENTS

SECTION Description .................................... PAGE

INTRODUCTION .................................................................................. 9
GETTING STARTED WITH THE EV80C196KB ........................................ 9
Powering the Board .................................................................. 9
Connecting to your PC .............................................................. 9
Starting the Host Software ........................................................ 9
HARDWARE OVERVIEW OF THE EV80C196KB BOARD ..................... 10
Block Diagram of the EV80C196KB Board .................................... 10
Processor .................................................................................. 10
Memory ..................................................................................... 10
Host Interface ........................................................................... 11
Digital I/O ................................................................................. 11
Analog Inputs ........................................................................... 11
Decoding ................................................................................... 12
Configuration Jumper Locations (Figure 3a) ................................ 14
Memory Configuration Jumper Locations (Figure 3b) .................... 15
Expansion Ports, Connectors and LEDs Locations (Figure 4) ........ 16
Host Serial Connector (Figure 5) ................................................ 17
80C196KB Serial Port Connector (Figure 6) ................................ 17
Analog Input Connector (Figure 7) .............................................. 18
I/O Expansion Connector (Figure 8) ............................................. 18
Memory-I/O Expansion Connector (Figure 9) ................................ 19
Power Supply Connector (Figure 10) ........................................... 19
25-pin to 9-pin Adapter (Figure 11) ............................................. 20
INTRODUCTION TO iRISM-iECM96 SOFTWARE .................................. 21
Features .................................................................................... 21
Restrictions ............................................................................... 22
OVERVIEW ........................................................................................ 23
Embedded Controller Monitor ..................................................... 23
USER INTERFACE ............................................................................. 24
Background Information ............................................................. 24
Initiating and Terminating iECM-96 .............................................. 25
Default Base Commands ............................................................. 28

FILE OPERATIONS .......... 29
Loading and Saving Object Code .......... 29
Other File Operations .......... 30
PROGRAM CONTROL .......... 32
Resetting the Target .......... 32
Breakpoints .......... 32
Program Execution .......... 33
Program Stepping .......... 35
DISPLAYING AND MODIFYING PROGRAM VARIABLES .......... 37
Supported Data Types .......... 37
BYTE Commands .......... 38
WORD Commands .......... 39
DWORD Commands .......... 40
REAL Commands .......... 41
STACK Commands .......... 42
STRING Commands .......... 42
Processor Variables .......... 43
ASSEMBLY AND DISASSEMBLY .......... 44
Single Line Assembly Commands .......... 44
Disassembly Commands .......... 45
SYMBOL OPERATIONS .......... 46
RISM .......... 47
RISM Variables .......... 47
RISM Structure .......... 48
Receiving Data from the Host .......... 48
Sending Data to the Host .......... 48
RISM Commands .......... 49

Schematics and Parts List .......... Appendix A

Specific iRISM Information .......... Appendix B

Listing of iRISM-196KB .......... Appendix C

Timing Analysis .......... Appendix D

Programmable Logic Equations .......... Appendix E

Standard Memory-I/O Connector .......... Appendix F

Sample Session .......... Appendix G

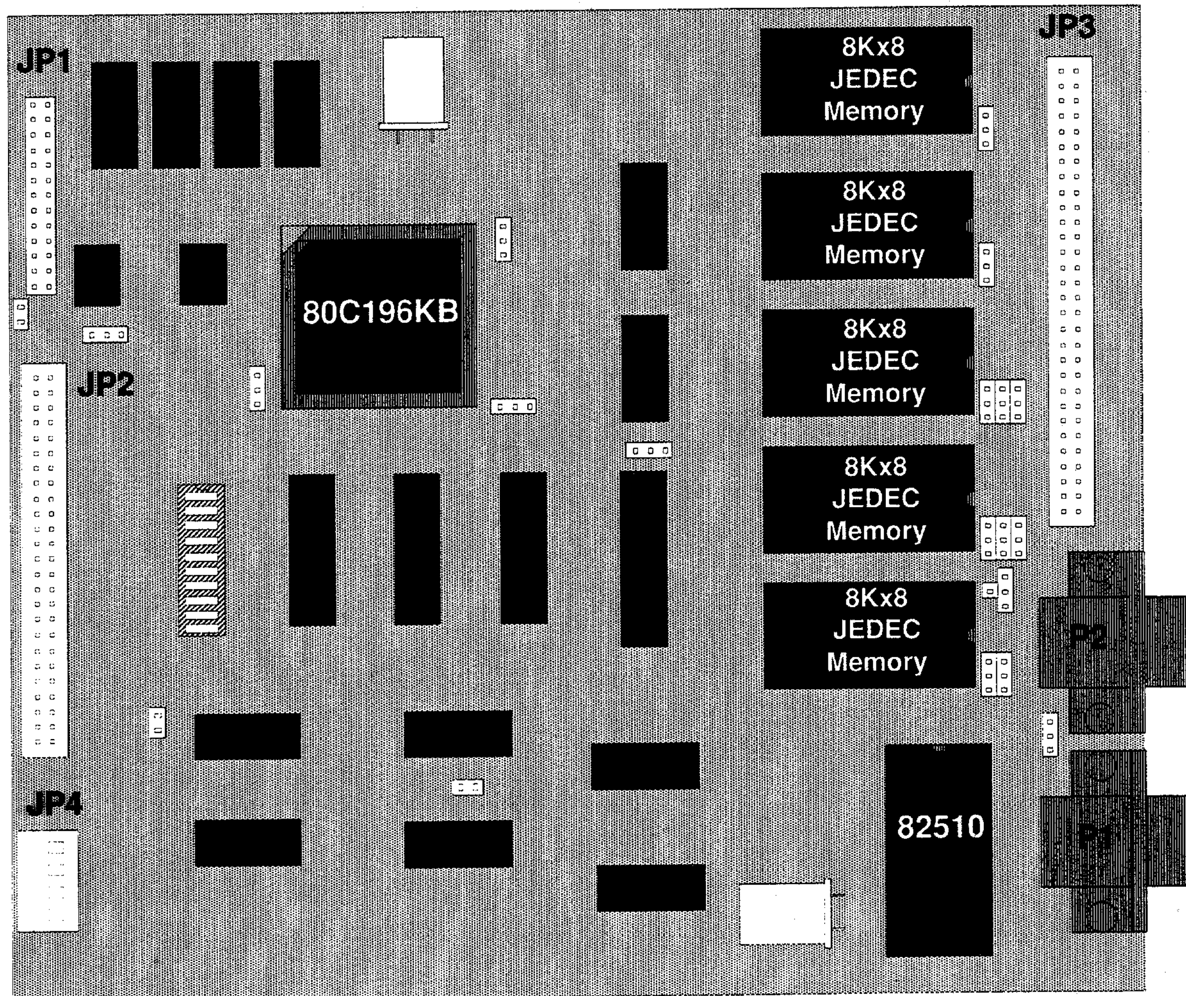



**Figure 1.**

**EV80C196KB Evaluation Board**

## INTRODUCTION

The EV80C196KB is a next-generation version of the EV80C196KA. The major changes are the use of a standard memory expansion bus compatible with the EV80C51FB and EV80C186 boards, and the removal of the card edge bus. Also, the HOLD/HLDA feature of the 80C196KB is supported. The EV80C196KB is designed to be a software evaluation tool for the ROMless 80C196KB 16-bit microcontroller. As such, ports 3 and 4 are not available for use as I/O ports unless offboard latches/buffers and decoding logic are used. All unreserved functions of the 80C196KB are available to you except for the Non-Maskable Interrupt (NMI), the TRAP instruction, and 512 bytes of address space. The Chip Configuration Byte is also used by the monitor, but most of its functions are provided by external logic.

## GETTING STARTED WITH THE EV80C196KB

### Powering up the Board

Power (+5, +/-12 Volts) must be connected to JP4 as shown on the board's silkscreen next to JP4 and in figure 10. Included with the board is a packet containing a Molex connector and crimp terminals for your convenience.

Power supply requirements for the EV80C196KB board are as follows:

| | |
|---|---|
| + 5 VDC +/- 5 % @ 280 mA | (150 mA if LED's are disabled by removing jumper shunt E16) |
| + 12 VDC +/- 20 % @ 15 mA | |
| - 12 VDC +/- 20 % @ 15 mA | |

Upon power-up (or after a reset) the board goes through initializations and a shifting-pattern is displayed on the Port 1 LEDs when initialization has completed properly.

### Connecting to your PC

Once you have applied power to the board, you need to connect P1 to a PC serial port. P1 is configured to interface pin-to-pin with a standard nine-pin AT(R)-type serial connector (see figure 5 for pinout). Make certain that you use a cable providing all nine signals, as they are all needed for proper operation of the host interface. When you have connected the cable, you may observe that the 80C196KB is held in reset, and all the LEDs turn on. This is because one of the host signals is used to reset the part, and the signal is often in a reset condition prior to invoking the host software on your PC.

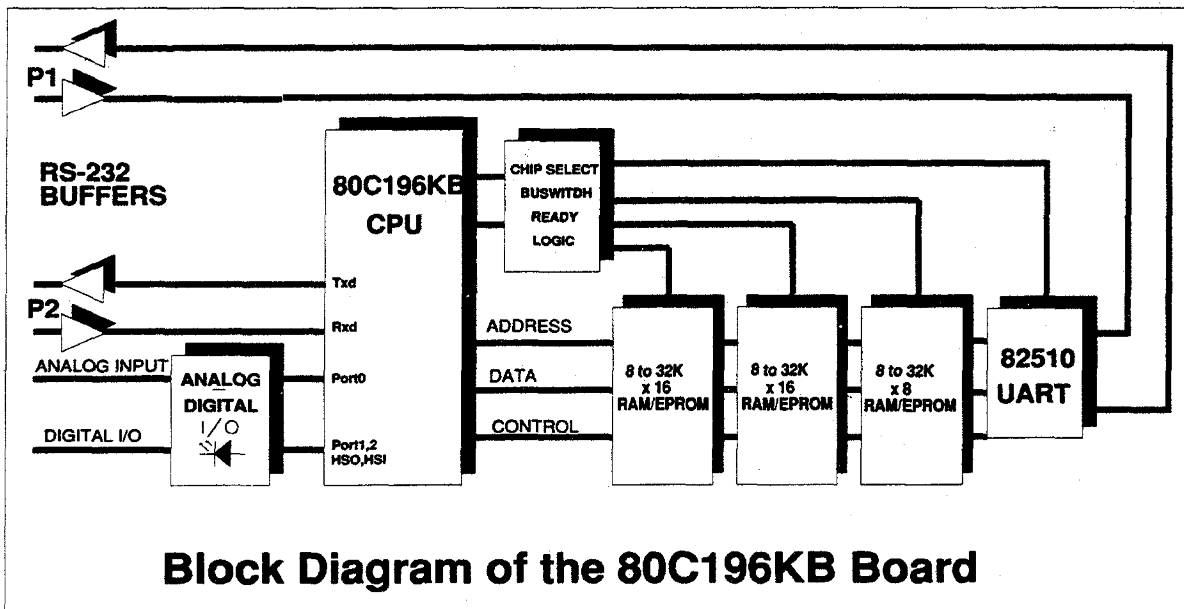
**Note: if you have a 25-pin serial port it will be necessary to make a 25-pin to 9-pin adaptor (see figure 11 for details).**

### Starting the Host Software

After the you have made both connections to the board, you can invoke the host interface. Install the disk in drive A of your system. At the DOS prompt type "A:ECM96"<CR>. Your PC should eventually display the iECM-96 monitor screen. If you have problems please refer to the sub-section "Initiating and Terminating iECM-96" in the "USER INTERFACE" section of this manual. For further details on using the monitor, refer to the "USER INTERFACE" section.

## HARDWARE OVERVIEW OF THE EV80C196KB BOARD

The EV80C196KB Microcontroller Evaluation board is delivered with an 80C196KB, 8 K-words and 8 K-bytes of user code/data memory, a UART for host communications and analog-input filtering with a precision voltage reference. Also included is programmable chip-select, bus-width and wait-state-counter logic which allows you to custom tailor the board to look like your own system. The board's physical dimensions are 6 1/2" x 7 3/4" with an overall height of 3/4". There are six main sections to the EV80C196KB board: Processor, Memory, Host Interface, Digital I/O, Analog Inputs and Decoding.




**Block Diagram of the 80C196KB Board**

**Figure 2.**

### Processor

The Intel[R] 80C196KB is a 16-bit embedded microcontroller. Being a member of the MCS[R]-96 family, the 80C196KB uses the same powerful instruction set and the same architecture as the existing MCS-96 products. The 80C196KB is an enhanced CMOS version of the 8097BH. Its enhancements include up/down and capture modes on Timer2, multiplying speeds almost 3 times as fast, overall execution nearly twice as fast, Hold/Hold Acknowledge logic, and power-down and idle modes to save power. For more information, please refer to the 1989 "16-Bit Embedded Controller Handbook," Intel Corporation order number 270646-001 and the 80C196KB Datasheet order number 270634-001.

### Memory

There are five 28-pin memory sockets provided on the EV80C196KB board: U1, U6, U8, U13 and U14. The sockets are designed to support byte-wide, JEDEC-pinout, memory devices of various types and sizes, i.e. 8K x 8 SRAM or 16K x 8 EPROM. U1 and U8, U6 and U13 are connected as two 16-bit memory banks and U14 is connected as an 8-bit memory bank.

| Bank No. | Even Bytes I.C. | Odd Bytes I.C. | Enable Signal | Memory Type |
|---|---|---|---|---|
| 0 | U8 | U1 | CE0 | 8K x 16-bit Monitor EPROM from 0-FFH and 1D00-1DFFH |
| 1 | U13 | U6 | CE1 | 8K x 16-bit ROMsim/RAM from 2000H-5FFFH |
| 2 | U14 | U14 | CE2 | 8K x 8-bit ROMsim/RAM from 6000H-7FFFH |

See appendix B and appendix C for details on reserved areas of memory.

### Host Interface

The PC host interface is accomplished with the 82510 UART (U20) connected to P1 via RS-232 drivers. The UART resides in the address range 1E00H - 1EFFH. Therefore, register 0 in the UART would be at address 1E00H of the 80C196KB, reg. 1 would be at 1E01H, reg. 2 would be at 1E02H, etc. up to reg. 7 at 1E07H. The registers will repeat again with reg. 0 at 1E08H due to the limited decoding granularity of the EPLD. Pin 12 of the UART, OUT1#, is used to tell the PC host when the 80C196KB is executing user code by a true level on the Ring Indicator input of the host serial port.

### Digital I/O

With the exception of the NMI input, which is used by the Host Interface, all Digital I/O functions of the 80C196KB are available to you. There are eight LEDs on-board along with buffer/drivers which allow you to quickly observe the state of Port 1, HSO.0 and Port 2.5/PWM (see figure 4 or the schematics in appendix A for location). The TxD and RxD pins of the 80C196KB (Port 2.0 and Port 2.1) are connected to RS-232 buffer/drivers, which are connected to P2. All of the I/O signals are available on JP2 (see figure 8 or the schematics in appendix A for pinout).

**Note: because RxD is connected to an RS-232 receiver (U19 pin 3) any attempt to use it as a digital input will result in a contention. If you would like to use it as a digital input, remove jumper shunt E19 to disconnect the receiver.**

### Analog Inputs

The Port 0 inputs of the 80C196KB double as both digital and analog inputs. The EV80C196KB board includes circuitry to make the analog inputs easier to use. A precision voltage source for Vref is provided on board (U3 and U4) which can be carefully adjusted by trimming RP1. Also, jumper shunt E4 allows Vref to be connected to Vcc instead of the output of U3. By removing E4 entirely, an off board reference can be connected to JP1. By removing jumper shunt E2, ANGND can be isolated from Vss. Protective clamping diodes are installed on each channel. RC networks are provided in sockets (to allow you to change the input impedance to match your application) on all of the analog input channels. If Port 0 is to be used

as a digital input, it is recommended that the capacitors be removed, and the resistors replaced with wires. For additional connection information refer to figure 7 or the schematics in appendix A. The ground and power planes beneath the analog circuitry (D1, D2, R3, C2, U3, U4, JP1 and the analog connections on the 80C196KB) are isolated from the digital power and ground planes of the board to keep noise from the analog inputs.

### Decoding

The decoding logic on the EV80C196KB board serves three purposes; to provide Chip-Enable signals to memory and peripheral devices, to select the buswidth for the device(s) being accessed and to provide wait-states for slow devices. This section is provided in case you need to modify the memory configuration of the EV80C196KB board. It is not necessary to understand this section for normal usage of the board.

The heart of the decoding logic is U12, a 24-pin 5AC312 Intel EPLD or a C22V10 programmable logic array which is socketed to allow easy changes. For the sake of convenience it will be referred to as "the EPLD" throughout this text. The EPLD uses latched addresses A8-A15 along with CLKOUT, HLDA#, RESET# and STALE (STretched ALE) from the 80C196KB as decode inputs.

There are 4 enable outputs from the EPLD, all of which are low-level true, however only one should be true at a time to avoid bus contention. They are decoded from the address lines, and an internally-latched signal called MAP. MAP is cleared when the RESET# input is true, and set when the Monitor EPROMs are accessed in the address range 1D00H-1DFFH. MAP will always be set when the board is in the USER mode.

| | |
|---|---|
| pin 21 = CE0 | Enables memory in U1 and U8<br>(monitor EPROM as shipped).<br><br>CE0 = (ADDRESS RANGE 2000H - 27FF and NOT MAP)<br>or ADDRESS RANGE 0H - FFH<br>or ADDRESS RANGE 1D00H - 1DFFH |
| pin 22 = CE1 | Enables memory in U6 and U13<br>(user 16-bit ROMsim/RAM as shipped).<br><br>CE1 = (ADDRESS RANGE 2000H - 27FFH and MAP)<br>or ADDRESS RANGE 2800H - 5FFFH |
| pin 15 - CE2 | Enables memory in U14<br>(user 8-bit ROMsim/RAM as shipped).<br><br>CE2 = ADDRESS RANGE 6000H - 7FFFH |
| pin 14 - CS510 | Enables U20, the 82510 UART, which is used for host communications.<br><br>CS510 = ADDRESS RANGE 1E00H - 1EFFH |

The BUSWIDTH output of the EPLD, pin 16, is fed into the buswidth pin of the 80C196KB. Therefore, it is driven low for accesses to 8-bit memory and high for accesses to 16-bit memory. As shipped, it goes low simultaneously with CE2 or CS510 as these are the only areas of memory mapped as 8-bit.

Programmed into the EPLD is a 3-bit wait-state machine clocked by the rising edge of CLKOUT from the 80C196KB. The transition sequence of the wait-state machine is controlled by the current state of the machine and the inputs to the EPLD (for further details see appendix E). While the bus of the 80C196KB is idle the wait-state machine is locked in state 0, which is called **async_start**. The conditions for leaving **async_start** are 1) ALE being asserted, 2) HLDA# not being asserted and 3) a value on A8 - A15 requiring wait-states. Because the falling edge of ALE can occur before the next rising edge of CLKOUT can clock the wait-state machine, a signal called STALE (for Stretched ALE) is used. STALE does not go low until after the rising edge of CLKOUT.

During **async_start**, the output WAIT# from the EPLD is asserted asynchronously based upon a value on A8-A15 requiring wait-states. If no wait-states are required, WAIT# will not be asserted and the wait-state machine will remain in **async_start**. However, if one or more wait-states are needed WAIT# will be asserted and the wait_state machine will transition out of **async_start** on the next rising edge of CLKOUT. The next state entered depends on how many wait-states are needed. If only one is required the next state is **remove_hold**, where WAIT# is deasserted regardless of the inputs to the EPLD. If two wait-states are needed the next state is **hold_2**, where WAIT# is always asserted, then the state after that is **remove_hold**. The additional states, **hold_3** - **hold_7**, work just like **hold_2** with WAIT# always asserted. The wait_state machine will count through from **hold_2** to **hold_n** to generate n wait-states before jumping to **remove_hold** to deassert WAIT#. The maximum number of wait-states is seven.

The previous paragraph described how the signal WAIT# is generated based on the rising edge of CLKOUT. However, the 80C196KB needs to have a valid signal on it's READY input pin until the falling edge of CLKOUT. Therefore, it was necessary to clock WAIT# through a negative-edge-triggered-JK flip-flop (U15A) by the falling edge of CLKOUT to generate a signal called WAITN#. As in the EPLD, WAITN# is asserted asynchronously while ALE is high and WAIT# is asserted. After ALE goes low WAITN# will remain asserted until WAIT# is deasserted and the flip-flop is clocked. Besides the WAIT# signal, the WAITN# signal can be asserted by the USEREADY signal from the expansion bus. As shipped, the EPLD has the following configuration:

| Memory Type | Wait States | Enable Signal | Memory Region in User Mode |
|---|---|---|---|
| ROMsim/RAM | 0 | CE1 | 2000H-5FFFH |
| ROMsim/RAM | 0 | CE2 | 6000H-7FFFH |
| Monitor EPROM | 1 | CE0 | 0-FFH, 1D00H-1DFFH |
| 82510 UART | 2 | CS510 | 1E00H-1EFFH |
| Unimplemented | 0 | N/A | 100H-1CFFH, C000H-FFFFH |
| Unimplemented | 1 | N/A | 8000H - BFFFH |

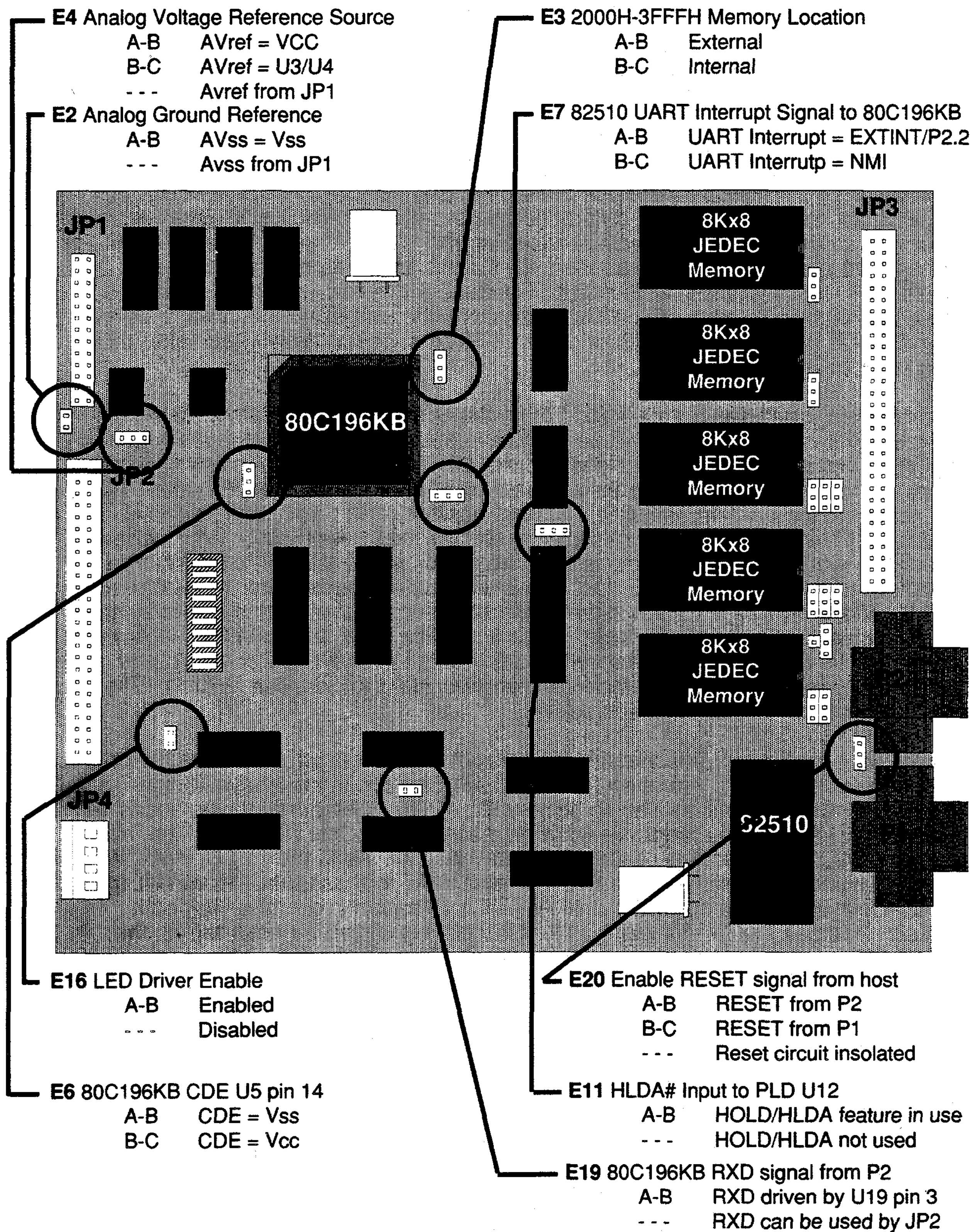



**Figure 3a.**
**Configuration Jumper Locations**

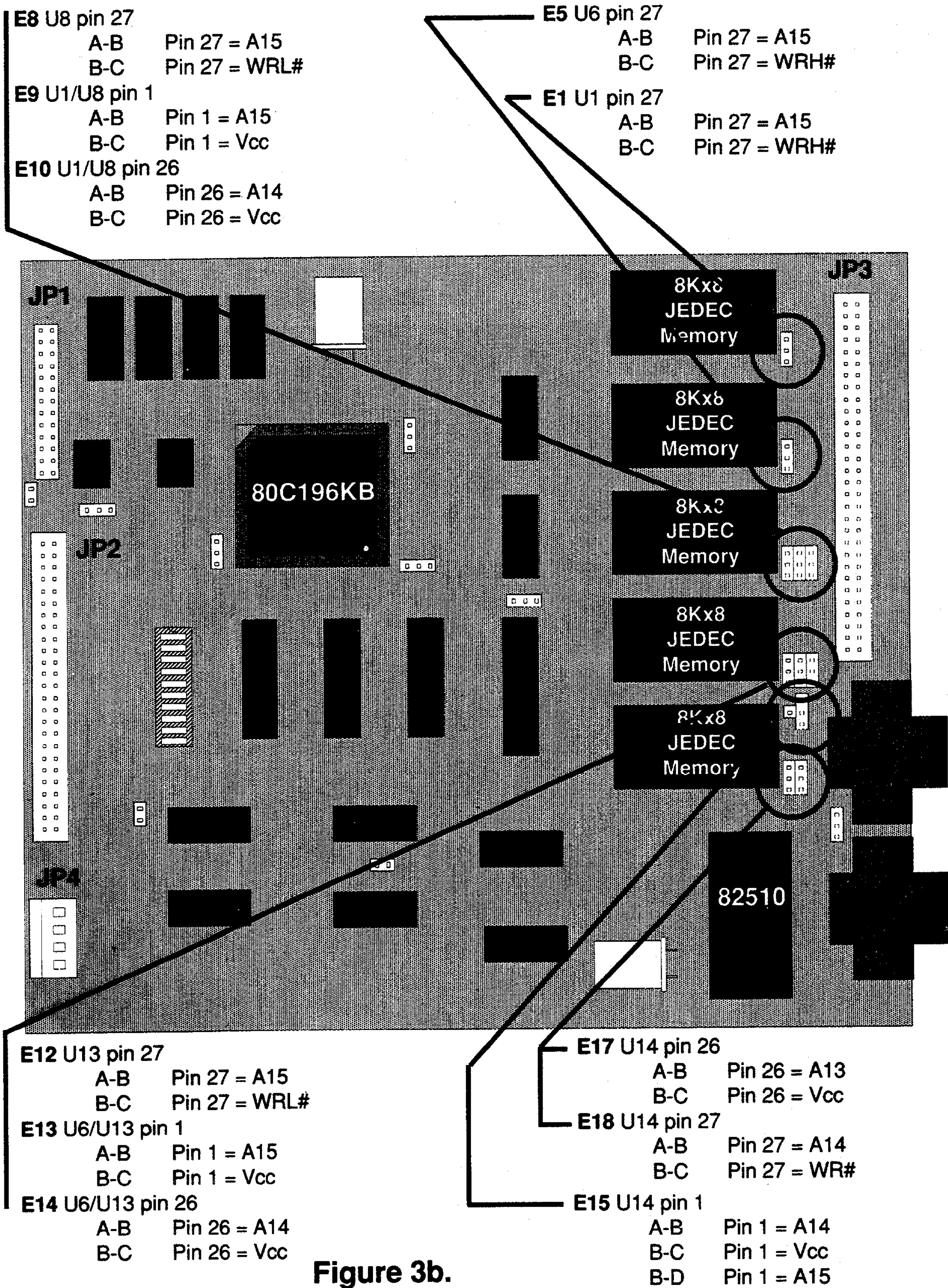



**Figure 3b.**
**Memory Configuration Jumper Locations**

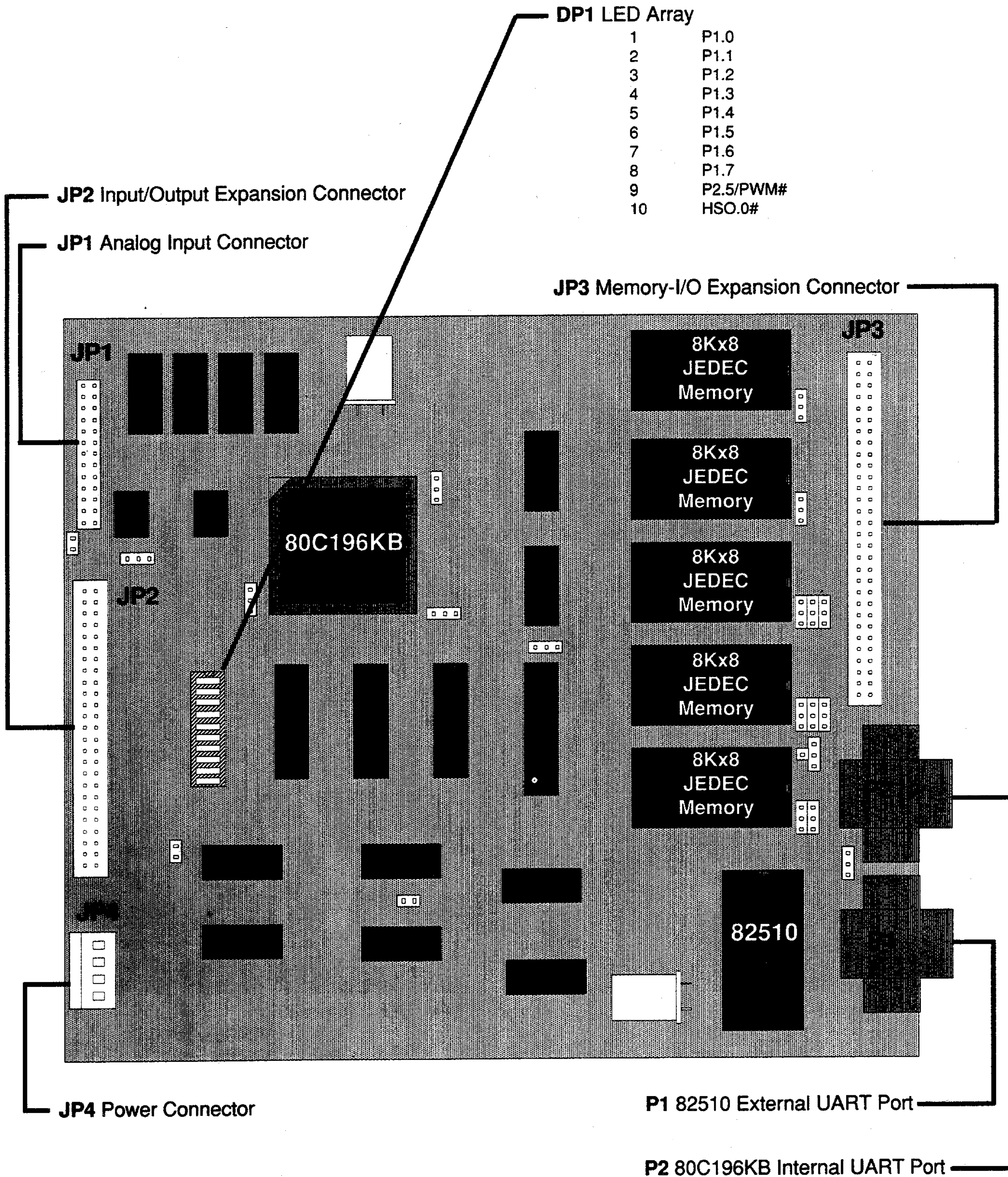
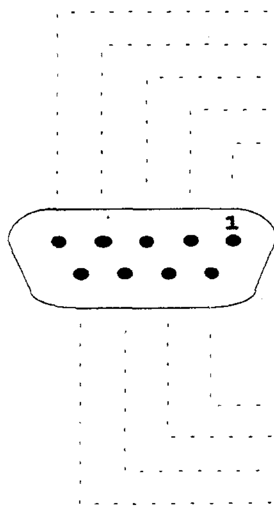



**Figure 4.**
**Expansion Ports, Connectors and LEDs**

## P1 Host Serial Connector
## DB-9S RS232

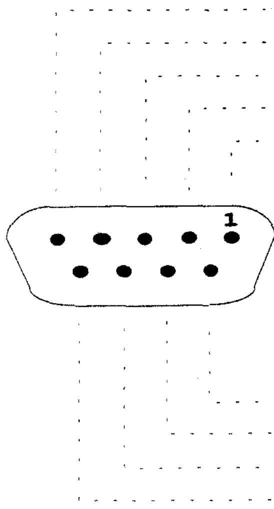



| Pin Nos. | Host RS-232 Signal Name | Connection on Evaluation Board |
|---|---|---|
| 5 (AB) | SG Signal Ground | Digital Ground |
| 4 (CD) | DTR Data Terminal Ready | INIT thru E20-C |
| 3 (BA) | TxD Transmit Data | RxD of 82510 |
| 2 (BB) | RxD Receive Data | TxD of 82510 |
| 1 (CF) | DCD Data Carrier Detect | DTR P1-pin 4 |

| Pin Nos. | Host RS-232 Signal Name | Connection on Evaluation Board |
|---|---|---|
| 6 (CC) | DSR Data Set Ready | DTR P1-pin 4 |
| 7 (CA) | RTS Request To Send | CTS P1-pin 8 |
| 8 (CB) | CTS Clear To Send | RTS P1-pin 7 |
| 9 (CE) | RI Ring Indicator | Run Indicator |

**Figure 5.**

## P2 Serial Port Connector
## DB-9S RS232




| Pin Nos. | Host RS-232 Signal Name | Connection on Evaluation Board |
|---|---|---|
| 5 (AB) | SG Signal Ground | Digital Ground |
| 4 (CD) | DTR Data Terminal Ready | INIT thru E20-A |
| 3 (BA) | TxD Transmit Data | RxD of 80C196KB |
| 2 (BB) | RxD Receive Data | TxD of 80C196KB |
| 1 (CF) | DCD Data Carrier Detect | DTR P2-pin 4 |

| Pin Nos. | Host RS-232 Signal Name | Connection on Evaluation Board |
|---|---|---|
| 6 (CC) | DSR Data Set Ready | DTR P2-pin 4 |
| 7 (CA) | RTS Request To Send | CTS P2-pin 8 |
| 8 (CB) | CTS Clear To Send | RTS P2-pin 7 |
| 9 (CE) | RI Ring Indicator | No connection |

**Figure 6.**

## JP1 Analog Input Connector

2x13 Pin MOLEX 39-51-2604 or Equiv.

| Odd pins | Even pins |
| --- | --- |
| ANGND - 1 | 2 - Analog Channel 0 |
| VREF ---- 3 | 4 - VREF |
| ANGND - 5 | 6 - Analog Channel 1 |
| ANGND - 7 | 8 - Analog Channel 2 |
| VREF ---- 9 | 10 - VREF |
| ANGND -11 | 12 - Analog Channel 3 |
| ANGND -13 | 14 - Analog Channel 4 |
| VREF --- 15 | 16 - VREF |
| ANGND -17 | 18 - Analog Channel 5 |
| ANGND -19 | 20 - Analog Channel 6 |
| VREF --- 21 | 22 - VREF |
| ANGND -23 | 24 - Analog Channel 7 |
| VREF --- 25 | 26 - ANGND |

**Figure 7.**

## JP2 I/O Expansion Connector

2x25 Pin MOLEX 39-51-5004 or Equiv.

1 thru 49 - VSS

2 - P1.0 Bi-directional
4 - P1.1 Bi-directional
6 - P1.2 Bi-directional
8 - P1.3 Bi-directional
10 - P1.4 Bi-directional
12 - P1.5/BREQ# Bi-directional
14 - P1.6/HLDA# Bi-directional
16 - P1.7/HOLD# Bi-directional
18 - P2.0/Txd Output
20 - P2.1/Rxd Bi-directional
22 - P2.2/Extint Input
24 - P2.3/T2CLK Input
26 - P2.4/T2RST Input
28 - P2.5/PWM Output
30 - P2.6/T2UPDN Bi-directional
32 - P2.7/T2Capture Bi-directional
34 - HSO.0 Output
36 - HSO.1 Output
38 - HSO.2 Output
40 - HSO.3 Output
42 - HSI.0 Input
44 - HSI.1 Input
46 - HSI.2/HSO.4 Bi-directional
48 - HSI.3/HSO.5 Bi-directional
50 - VCC

**Figure 8.**

## JP3 Memory-I/O Expansion Connector

2x30 Pin MOLEX 39-51-6004 or Equiv.

| Signal | Pin | Pin | Signal |
|---|---|---|---|
| Vcc | 1 | 2 | Vcc |
| A0 Output | 3 | 4 | D0 Bi-directional |
| A1 Output | 5 | 6 | D1 Bi-directional |
| A2 Output | 7 | 8 | D2 Bi-directional |
| A3 Output | 9 | 10 | D3 Bi-directional |
| A4 Output | 11 | 12 | D4 Bi--directional |
| A5 Output | 13 | 14 | D5 Bi-directional |
| A6 Output | 15 | 16 | D6 Bi-directional |
| A7 Output | 17 | 18 | D7 Bi-directional |
| Vss | 19 | 20 | Vss |
| A8 Output | 21 | 22 | D8 Bi-directional |
| A9 Output | 23 | 24 | D9 Bi-directional |
| A10 Output | 25 | 26 | D10 Bi-directioal |
| A11 Output | 27 | 28 | D11 Bi-directional |
| A12 Output | 29 | 30 | D12 Bi-directional |
| A13 Output | 31 | 32 | D13 Bi-directional |
| A14 Output | 33 | 34 | D14 Bi-directional |
| A15 Output | 35 | 36 | D15 Bi-directional |
| Vss | 37 | 38 | Vss |
| CLKOUT Output | 39 | 40 | Vss |
| RD# Output | 41 | 42 | WR# Output |
| BREQ# Output | 43 | 44 | BHE# Output |
| ALE Output | 45 | 46 | UserReady Input |
| NMI Input | 47 | 48 | INST Output |
| RESET# Output | 49 | 50 | P2.2/EXINT Bi-directional |
| No Connection | 51 | 52 | No Connection |
| HLDA# Output | 53 | 54 | HOLD# Input |
| -12VDC | 55 | 56 | +12VDC |
| Vss | 57 | 58 | Vss |
| Vcc | 59 | 60 | Vcc |

**Figure 9.**

## JP4 Power Supply Connector

4 Pin MOLEX 26-03-3041 or Equiv.

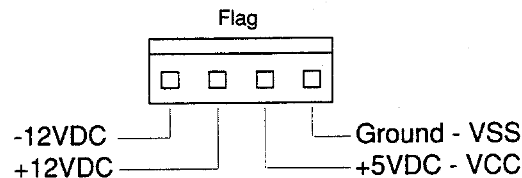



**Figure 10.**

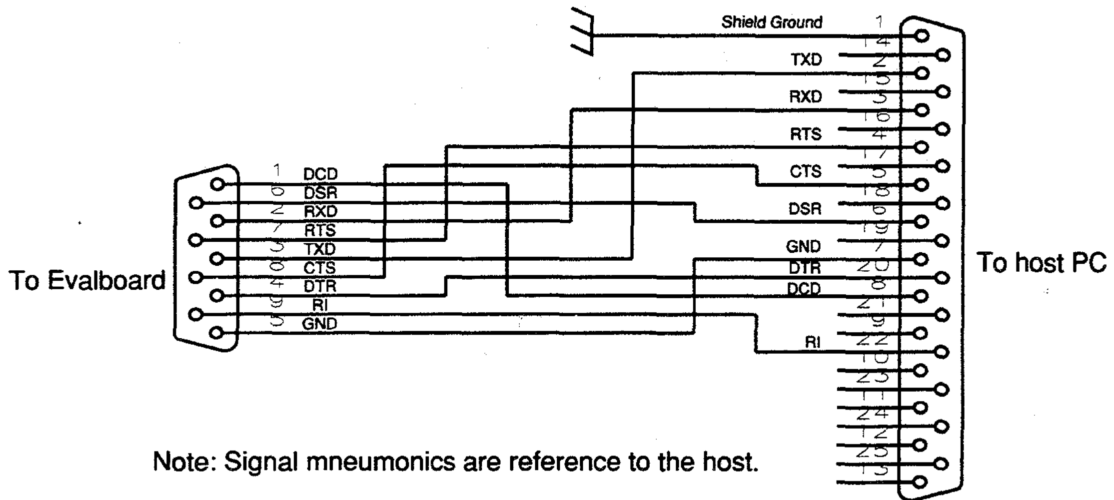



**Figure 11.**
**25-pin-to-9-pin Adapter**

## INTRODUCTION TO iRISM-iECM SOFTWARE

The EV80C196KB board uses an Embedded Controller Monitor (ECM) written for the MCS-96 family of 16-bit microcontrollers. This monitor supports basic debug facilities (LOAD, GO, STEP etc.) in the user's target system. The ECM is broken into two independent programs, one of these executes in the EV80C196KB (iRISM-96KB) and the other executes in a IBM PC or BIOS compatible clone(iECM-96). These two programs communicate via an asynchronous serial channel using a binary protocol defined specifically for this application.

The partitioning of the ECM into two separate programs supports a number of goals in the development of this system:

The system is easy to adapt to a new target because the code which runs in the target is very simple and small.

The feature set of the user interface is not limited by the resources of the target since the user interface is implemented in the host PC.

Concurrent operation of the ECM and the target system was easily achieved. This allows you to interrogate and (carefully) modify the state of the target system while it is running.

This manual section describes the user interface provided by the iECM-96, the interface between this PC resident software and the target resident software, and the structure of the software in the target. Appendix B lists the resources of the 80C196KB that are reserved for this RISM implementation. Appendix C is the listing for the iRISM software which runs in the 80C196KB on this board. It uses an Intel 82510 UART for host communications.

The iECM-96 was designed and implemented by Intel to support user's of the MCS-96 architecture, and is placed in the public domain with no restrictions or warranties of any kind.

### Features

Host system is an IBM PC AT, PC XT, or BIOS-compatible clone. (Interfaces via COM1 or COM2 at 9600 baud.)

Sixteen software execution breakpoints

Concurrent interrogation of target memory and registers

Supports BYTE, CHARACTER, WORD, STRING, DOUBLE-WORD and FPAL-96 REAL variable types.

Single-Line Assembler/Disassembler

Symbolics compatible with Intel's OMF debug records

Supports LOAD, SAVE, LIST, LOG, and command INCLUDE files.

### Restrictions

Two words of user stack are reserved for use by the iRISM-96 software.Other memory and/or registers in the target memory will be used by the iRISM-96 software. The exact number and location of this memory is implementation dependent. See appendix B or C for further information.

An asynchronous serial port capable of operation at 9600 baud must be available in the target system. The RISM described in this document uses an Intel 82510 UART. This version also uses the NMI (Non-Maskable Interrupt) to signal that a received data character is available.

The TRAP instruction is reserved.

Breakpoints and program stepping will not operate if the user's code is in EPROM or other nonchangeable memory.

## OVERVIEW

### Embedded Controller Monitor (ECM)

An ECM (Embedded Controller Monitor) provides basic debug capability and is installed in your target system. Capabilities include loading object files into system RAM, examining and modifying variables, executing code, and stepping through code. In the past, most of these monitors have been configured to run with a standard "dumb" CRT with some form of auxiliary port for loading and saving object code from a host system. It is now common for a personal computer to act as the host for program translation and also emulate a dumb CRT during user interaction with the ECM. The ECM developed for the MCS-96 family makes the assumption that the user interface will always be a personal computer; no provision is made for interface to a dumb CRT. By making this assumption it is possible to reduce the size and complexity of the code that must be installed in the target system. A term has been coined for this code resident in the target -- RISM. The term RISM stands for Reduced Instruction Set Monitor and is an obvious takeoff of the term RISC (Reduced Instruction Set Computer) used to describe a class of computer architectures. The RISM consists of about 300 bytes of MCS-96 code which provide primitive operations. Software running in the host uses the RISM commands to provide a complete user interface to the target system. The advantage of this approach is that the ECM can be readily adapted to different target systems and requires only a small part of the available target memory space. The disadvantage is that the user interface must be provided by a personal computer.

The structure of the RISM is a short section of initialization code and an interrupt service routine (ISR) that processes interrupts from the host system. The RISM ISR consists of a short prologue and then a case-jump to one of 20 to 25 command executors. These executors are simple and short; the flow though the entire ISR (including the prologue) is 15-20 instructions. The serial communication occurs at 9600 baud, which limits the frequency of these interrupts to 1 Khz. In the worst case the EV80C196KB board will be slowed by the execution of a fairly short RISM ISR every millisecond while executing user code. It is possible to operate the EV80C196KB board so that no real-time is lost to the iECM-96 unless the user is actively interrogating the target. (See the section "Initiating and Terminating the iECM-96" and the description of the RISM REPORT_STATUS command for details on this).

## USER INTERFACE

The user interface to the iECM-96 supports commands to initiate and configure the ECM-96, perform I/O operations involving DOS files, execute user programs, and interrogate variables in the target system. Interrogation can be done in a number of formats and in most cases can be done concurrently with user code execution. A single line assembler and disassembler are also provided.

**Note: on the disk included with the EV80C196KB is a file called DEMO.LOG. DEMO.LOG is a sample iECM-96 session for you to invoke and become more familiar with the features of iECM-96. Appendix G is a printout of DEMO.LST which was created by turning on the list feature and invoking DEMO.LOG by typing "include demo.log"<CR> at the iECM-96 "*" prompt.**

### Background Information

Numeric and Symbolic Input
The command parser used by the iECM-96 software requires that numeric inputs always start with the digits 0-9. If hexadecimal numbers are entered which start with A-F they must be preceded by a "0". For example, enter "0AA55" instead of "AA55". This requirement is similar to ASM-96. If symbolic information has been downloaded as part of an object file (see "Loading and Saving Object Code") then you can enter a valid symbol name whenever a number is expected. The symbol name must be preceded by a period (".") so that the parser knows to try searching the symbol table. If the symbol is ambiguous then it will not be accepted by the parser. The probability of ambiguous references can be reduced by specifying the module name along with the symbol name. The module name must be preceded with a colon (":"). If a variable TEMP is declared both in MODULE1 and in MODULE2, then a reference to the TEMP declared by MODULE1 would be ":MODULE1.TEMP". PLM-96 or C-96 line numbers can be called out by a pound sign ("#") followed by the line number.

Symbolic Output
The symbolic output routines, in general, deal only with address information. They will not try to convert data values into symbolic form. When the symbol table is searched for a symbol name to associate with a given value the routines also perform type checking. If one, and only one, symbol matches both the type and value of the address being displayed then the output routines will display the symbol name along with the numeric value of the address. If more than one label has been assigned to a given address then the symbolic output routines will ignore all of them. The exception to this rule occurs when the disassembler finds multiple labels assigned to a given code address. The disassembler will display all the known symbolic labels attached to a code address.

If the symbols table gets very large the symbolic output routines will become painfully slow, particularly on an 8088 based PC. This problem can be avoided by using modular programming and translating a subset of the modules in the debug mode. Another alternative is to use the "SYMBOLS OFF" command to suppress symbolic output. Symbolic input is not affected by this command.

Controlling Lengthy Commands
Most of the commands supported by iECM-96 appear to complete without delay. Some commands (e.g. displaying or filling a large area of memory) take an appreciable length of time to complete. In general these commands can be aborted by entering a CARRIAGE-RETURN. Those commands which display a large amount of information can be paused by hitting the SPACE bar. After you have checked the data currently on the screen you can depress the SPACE bar again to resume the output.

Aborting from iECM-96
Entering a control-C will cause the iECM-96 to close any open files and return to DOS.

**Initiating and Terminating iECM-96**

This section describes the commands for invoking iECM-96 from DOS and exiting back to DOS.

ECM96
This command, entered at the DOS prompt, loads the iECM-96 software and executes it. Several options are available with this command. Option strings always start with a hyphen ("-") and can be entered in upper or lower case. The operation of these options is described below. Any or all of these options can be entered in any order, if the options are contradictory then the actual option accepted is the last one entered.

-COM2, -COM1
These options tell the iECM-96 software which serial communication port is to be used. If neither of these options is entered then COM1 will be used as a default. If iECM-96 detects valid CTS (Clear To Send) and DSR (Data Set Ready) signals from the appropriate COM port it will sign on and display a command prompt. If the target is stopped the command prompt will be an asterisk ("*"). If the target is already running the prompt will be a greater-than sign (">").

-DIAG
If CTS or DSR are not present, iECM-96 will complain about it and ask if you want to proceed or exit. It is possible, but not likely, that iECM-96 will operate properly even after complaining. It is more likely that there is a problem with the serial port or the cabling which will prevent proper operation. If the problem is not obvious (e.g. disconnected cable or no power to the target hardware) then the -DIAG invocation option can be used to help isolate the problem. The -DIAG option puts the iECM-96 system in a special mode which allows many tests to be used to find interfacing problems, or target bugs.

The diagnostic mode is intended to support debugging of boards which use the iECM-96. It can be particularly useful in systems which have multiple address decoding modes, such as the EV80C196KB. Upon reset this board has EPROM at location 2080H, the address where the 80C196KB starts execution. After executing some initialization code, the board can change the address decoding so that ROMsim/RAM is available in the partition which contains 2080H and the RISM is relocated to another area. This allows you to download code which is designed to operate in the on-chip ROM of MCS-96 family parts (2000H - 3FFFH). The diagnostic mode allows the use of diagnostic routines which disappear from memory space

when the RAM is mapped into the system. It also provides a simple routine to check the communications interface between the host and the target.

In the EV80C196KB board, there is a serial port loop-back mode which allows debugging the host/board interface. Upon reset the board is in the echo mode. Until it receives an ASCII slash ("/") or reverse-slash ("\") it will increment every character it receives from the host and send the incremented value back to the host. It will also display the binary code of the character the board received on the Port 1 LED's. If a reverse slash is received by the RISM it will leave the echo mode (set USER_MAP flag true), remap memory and start normal operation. If a slash is received it will stop echoing incremented received data and start responding to RISM commands with the diagnostic flag set. In this mode there are diagnostic routines resident in EPROM which are useful for debugging the board. Initially after invoking the diagnostic mode, the Program Counter points to the beginning of a RAM test at 2200H. See the source code listing in appendix C for further details.

**Note: The target hardware will have to be reset before using the DIAG command option.**

**Note: When executing diagnostic routines from EPROM, certain commands such as Breakpoints and Stepping will not work as they need to modify the code to work properly.**

When the host software is invoked in the diagnostic mode it will tell you to enter characters on the keyboard. These characters will be sent to the target and the response from the target will be displayed on screen. This is a simple confidence check on the serial communication channel. You are told to enter a slash or reverse-slash to terminate this mode and proceed in either the diagnostic mode or the normal user's mode. If the user interface is invoked without the -DIAG option it will immediately transmit a reverse-slash which should put the target in the normal mode. Systems which do not implement the diagnostic mode will load the reverse-slash into the RISM_DATA register where it will languish till more useful data is sent by the host.

-8096, -8096BH, -C196KB
These three options control the single line assembler and the disassembler in the iECM-96. If the 8096 (8x9x-90) or 8096BH (8x9xBH) options are selected then the additional instructions in the 80C196KB will be considered invalid for both the single line assembler and the disassembler. If none of these options are selected then the iECM-96 will default to C196KB mode.

-NOTYPES
This option will cause the object file loader to ignore type definition records in the object module. If this is invoked then the symbolic I/O routines will only recognize basic data types such as BYTEs, WORDs, and LONGs. More complex data types such as PLM arrays and structures will not be recognized. This option is included because early versions of the host software got confused while loading certain type definition records generated by C-96. These problems have been fixed but the option was left in case similar problems remain.

-POLL, -SIGNAL
These two options control how the host software detects whether or not the user's code is running. If poll mode is selected then the host will periodically poll the target with a REPORT_STATUS command. This takes no additional hardware but forces the target to waste instruction cycles responding to the poll. The signaling mode avoids this overhead but requires that the target set the Ring Indicator modem control line whenever it is running user code. The user interface will then check this line before it issues a REPORT_STATUS command. If neither of these options is selected then the signal mode is selected as a default. On the EV80C196KB the OUT1# pin of the 82510 is used to generate this running signal. Therefore, the signal mode is recommend.

RESET SYSTEM
RES SYSTEM
RESET
RES
This command and its abbreviations will reset the entire target hardware system if the target system is implemented to support this operation. On the EV80C196KB jumper shunt E20 must be installed from B to C for this command to work properly. This command operates by dropping the DTR modem control line. This comes into the target as DSR. After dropping DTR the iECM-96 software will wait about 1 second to allow the target to complete its initialization routines. The iECM-96 will politely warn of this time delay and then ignore the user until it expires. Unless special precautions are taken in the design of the target system, any data in RAM (including downloaded object code) may be corrupted by the reset. On the EV80C196KB, the RAM contents should not be affected by a RESET.

DOS
This command enables you to temporarily leave iECM-96 and return to DOS. Once you have suspended iECM, you may perform other functions in DOS, including using other software programs, such as ASM-96, as long as there is sufficient memory to do so.

To reenter iECM, type **exit** at the DOS prompt. iECM will return with all conditions in effect at the time it was suspended.

QUIT
This command will close any files that iECM-96 has opened and exit to DOS. Note that this command can be used even if the target is running. iECM-96 sets the selected COM port to 9600 baud, 8 bits, no parity, and one STOP bit. The port will be left in this state by iECM-96 when control is returned to DOS.

### Default Base Commands

These commands are used to set the default base for numeric input and output. The valid bases are: 16 (hexadecimal) , 10 (decimal), and 8 (octal). The default base is used to display variables. It is not used to display addresses (which are displayed in hexadecimal) or breakpoint numbers (which are displayed in decimal). The default base is also used to enter numbers into the command parser, but it is possible to override the default base during input by adding a character at the end of the number which forces the appropriate base to be used. The override characters are H (or h) for hexadecimal, T (or t) for decimal, and O (or o) for octal. The override character must appear immediately following the last digit of the number with no intervening space.

BASE
This command will display the current default base.

BASE=<valid_base>
This command will set the current default base to <valid_base>. When entering this command it is advisable to use an override character to select the new default base:

```
BASE=10O   ; selects octal
BASE=10T   ; selects decimal
BASE=10H   ; selects hexadecimal
```

This avoids confusion when changing bases. As an example of the confusion which is avoided, consider the following commands entered while the base is hexadecimal. The command:

```
BASE=10
```

will leave the default base as hexadecimal and the command:

```
BASE=16
```

will result in an error because 16H (22T) is not a valid base. The command:

```
BASE=0A
```

will select decimal as the default base but it is cleaner and simpler to use the override character:

```
BASE=10T
```

This works independently of the current default base and leaves a useful record in log or list files which may be open.

## FILE OPERATIONS

iECM-96 uses files in the host system to load and save object code, enter predefined strings of commands, to keep a log of commands that are entered by the user, and to keep a record of an entire debug session which includes both the characters entered by the user and the response generated by iECM-96 on the host screen. The commands which operate with files are described in the following sections.

### Loading and Saving Object Code

iECM-96 accepts object files which are generated by Intel's development tools. iECM-96 will not accept files which contain unresolved externals or files which contain relocatable records. These files must be passed through RL-96 in order to resolve the externals and/or absolutely locate the relocatable segments. iECM-96 will also not accept HEX format files. There is a utility on the disk (HEXOBJ.EXE) for converting HEX format files to Intel object format files loadable by iECM-96. While still in DOS type "HEXOBJ <filename>.hex <filename>.obj"<CR> to convert <filename>.hex to a usable format for iECM-96. HEXOBJ does not attempt to convert any symbolic information contained in the HEX file. The iECM-96 commands which operate on object files are:

LOAD <filename>
LOADSYM <filename>
SAVE <addr> TO <addr> IN <filename>

The metasymbol <filename> means that a valid MS-DOS file name must be entered in that position of the command string.

LOAD <filename>
This command loads the content records of the object file <filename> into the target memory and loads any associated symbolic information into a symbol table maintained in the host system's memory.

LOADSYM <filename>
This command loads the symbolic information from <filename> into the symbol table maintained in the host system but does not load the content records into the target's memory. This command is useful when you have left a debug session with the target still running a program that has been loaded. At a later time you can re-invoke iECM-96 and interrogate the running program without stopping it. The LOADSYM command allows the use of the symbolic information contained in the object file without reloading the content records. (Content records cannot be loaded while the target is running).

SAVE <addr> TO <addr> IN <filename>
This command saves a region of memory as an object file which can be reloaded into the target memory at some latter time. No attempt is made to include any symbolic information which may have been in the symbol table maintained in the host system.

### Other File Operations

In addition to object files, the iECM-96 makes use of include files, log files, and list files. Include files contain commands to be executed by iECM-96, they must contain the exact sequence of ASCII characters that you would enter from the keyboard to execute the command. Include files can be tedious to generate with a text editor so iECM-96 can generate log files in which are stored characters entered by the user. The intent is that log files be used later as include files to recreate command sequences. List files keep a running record of both commands entered by the user and of the response generated by iECM-96. Comments can be included in list and log files to make them easier to understand. A comment starts with a semicolon (';') and ends with a carriage return or ESC. The semicolon is considered to be part of the comment but not the CR or ESC. The command parser will ignore comments but will put them in the list and log files.

**Note: on the software disk included with the EV80C196KB is a file called DEMO.LOG. DEMO.LOG is a sample iECM-96 session for you to invoke and become more familiar with the features of iECM-96. Appendix G is a printout of DEMO.LST which was created by turning on the list feature and invoking DEMO.LOG by typing "include demo.log"<CR> at the iECM-96 "*" prompt.**

The list and log files commands allow for default filenames and allow either overwriting existing data in the file or appending data at the end of the file. This allows you to gather list and log data in the default files which avoids the creation and management of a large number of separate files. Log and list files are stamped with the date and time whenever they are opened to make it easier to use this capability and then go back and sort out the data from several debug sessions with a text editor.

The commands involved in include, log, and list operations are:

INCLUDE <filename><br>
PAUSE LIST<br>
LIST <filename><br>
LOG<br>
LOG <filename><br>
LISTOFF<br>
LISTON<br>
LOGOFF<br>
LOGON

Three of these commands require you to supply a valid file name, the rest use the appropriate file name that has already been entered.

INCLUDE <filename><br>
This command will attempt to open <filename> as a read only file. If the file can be opened then the command parser will take commands from that file until the end of the file is reached. The include file will then be closed. Only one include file will be opened at a time.

PAUSE
This command is documented in this section because it is intended to be used as part of INCLUDE files. It is not really a file oriented command itself. When this command is entered the iECM-96 will stop parsing commands until a SPACE character is entered from the keyboard (it can't come from an INCLUDE file). This provides a method of pausing in the middle of an INCLUDE file operation until you have a chance to see what's going on and acknowledge the pause condition by depressing the SPACE bar.

LIST
This command behaves like the LIST <filename> command described below except that it uses the last <filename> that was entered as part of a LIST <filename> command. If no such command has been entered then the default filename "LIST.ECM" will be used.

LIST <filename>
This command will attempt to open <filename> as a writable file. If a file with <filename> already exists then iECM-96 will ask if the file is to be overwritten or if the new data should be appended to the end of the existing file. It will then open the file and stamp it with the current date and time from the system clock. After this, commands entered by the user and the responses generated by iECM-96 will be recorded in the file.

LOG
This command behaves like the LOG <filename> command described below except that it uses the last <filename> that was entered as part of a LOG <filename> command. If no such command has been entered then the default filename "LOG.ECM" will be used.

LOG <filename>
This command will attempt to open <filename> as a writable file. If a file with <filename> already exists then iECM-96 will ask if the file is to be overwritten or if the new data should be appended to the end of the file. It will then open the file and stamp it with the current date and time. After this, commands entered by the user will be recorded in the file. Note that this file may contain nonprintable characters (e.g. ESC).

LISTOFF and LISTON
The LISTOFF closes a LIST file that has been specified by the LIST command. This stops new list information from being recorded. The LISTON re-opens the list file in the append mode so that recording can start again. LISTON also stamps the list file with the current date and time from the system clock.

LOGOFF and LOGON
The LOGOFF closes a log file that has been specified by the LOG command. This stops new list information from being recorded. The LOGON re-opens the log file in the append mode so that recording can start again. LOGON also stamps the list file with the current date and time from the system clock.

## PROGRAM CONTROL

Commands which control program execution allow you to reset the processor, set execution breakpoints, start execution, stop execution, step, and super step. The commands will be grouped by their major function for the sake of discussion.

### Resetting the Target

The processor can be reset by executing the iECM-96 command:

RESET CHIP
This command physically resets the processor by setting the RISM_DATA register to 0XXXX0001 and issuing a MONITOR_ESC RISM command which will cause the target to perform a RST instruction.

### Breakpoints

iECM-96 provides sixteen program execution breakpoints. If a given breakpoint is inactive it is set to zero, if it is active then it is set to the address of the first byte of an instruction. Breakpoints set to addresses which are not the first byte of an instruction will cause unpredictable errors in the execution of the user's code. When execution is started iECM-96 saves the user code byte at any active breakpoint and substitutes a TRAP instruction for that byte. Executing a TRAP instruction will cause the iECM-96 to restore the user code bytes where the TRAP instructions were substituted and then decrement the user's program counter so that it points at the original instruction. The user's program will appear to stop execution immediately before executing the instruction with a breakpoint set on it. All the TRAPs will be removed from the user's code and the original code restored.

**Note: Most monitor programs similar to iECM-96 display a message on the console when a break occurs (e.g. "Program break at 1234H"). This is not done in iECM-96 because the system supports concurrent interrogation of the target which the user's code is running; it is possible (perhaps probable) that the break will occur while you are in the middle of displaying or modifying the state of the target. Any special break message would have to interrupt the execution of the command. Because of this the iECM-96 does not output a special break message. You have two ways to find out that a break occurred:**

1). The prompt will change from a greater-than ">" to an asterisk ("*").

2). The status of the processor shown in the "control panel" at the top of the console screen will change from "running" to "stopped".

Commands which set the breakpoint array are:

```
BR
BR [ <bp_number> ]
BR [ <bp_number> ] = <code_addr>
```

The square brackets in the latter two commands are part of the command syntax and must be entered by the user, the angle brackets are part of the "meta" language used to describe the syntax. Breakpoints can be displayed while your code is running but they cannot be modified.

**NOTE: BR[0] and BR[1] can also be set by the GO command by using the TILL clause; all of the breakpoints will be cleared by the GO command if the FOR-EVER clause is used.**

BR
This command will display all of the active breakpoints (i.e. those not set to zero). You will also be informed if no breakpoints are active.

BR [ <bp_number> ]
This command will display the setting of the selected breakpoint and wait for input from you. If you enter a carriage-return the command will terminate. If you enter an ESC the next sequential breakpoint will be displayed. If you enter a numeric value then the selected breakpoint will be loaded with the value and the iECM-96 will again wait for input. At this point you can enter either a CARRIAGE-RETURN or an ESC. As before, the ESC will cause the iECM-96 to display the next breakpoint and the CARRIAGE-RETURN will terminate the command. This command will wrap around from the last breakpoint (15t) to the first breakpoint (0).

BR [ <bp_number> ] = <code_addr>
This command sets the specific breakpoint specified by <bp_number> to the value <code_addr>.

**Program Execution**

These commands start and stop execution of user code. The commands provided are:

GO
GO FOREVER
GO FROM <code_addr>
GO FROM <code_addr> FOREVER
GO FROM <code_addr> TILL <code_addr>
GO FROM <code_addr> TILL <code_addr> OR <code_addr>
GO TILL <code_addr>
GO TILL <code_addr> OR <code_addr>
HALT

If a GO with breakpoint command is entered, the user code bytes at the breakpoints will be saved and TRAPs will be installed. When a breakpoint is reached the user's software will stop *before* the instruction which caused the breakpoint and the iECM-96 software will restore the original user code. Note that this is different from the operation of iSBE-96 (and most ICE modules) which stop just *after* the instruction executes. A problem associated with stopping before the break instruction executes is that subsequent GO commands may run into the breakpoint before any user code is executed. The iECM-96 avoids this problem by skipping the setting of any break-points set on the instruction that the current PC points to. If this happens to remove the last breakpoint set then you will be warned but the GO will still execute with no breakpoints enabled. IF this happens you can use the HALT command to stop the program.

None of the GO commands can be executed while the user's code is already running; the HALT command cannot be executed if the user's code is not running. The GO commands which set breakpoints use BP[0] and possibly BP[1]. Any break value already in one of these breakpoints will be overwritten and destroyed by these GO commands. If possible the user should reserve the first two breakpoints for use by the GO commands and set the remaining breakpoints (if required) explicitly with the BR commands.

GO
This command starts execution of the user's code using the current value of user's PC and the current breakpoint array.

GO FOREVER
This command clears the breakpoint array and starts execution at the current value of the user's PC.

GO FROM <code_addr>
This command loads the user's PC with <code_addr> and starts execution of the user's code using the current breakpoint array.

GO FROM <code_addr> FOREVER
This command loads the user's PC with <code_addr>, clears the breakpoint array, and starts execution of the user's code.

GO FROM <code_addr> TILL <code_addr>
This command loads the user's PC with the <code_addr> which follows the FROM keyword, sets the first breakpoint (BP[0]) to the <code_addr> which follows the TILL keyword, and then starts execution of the user's code.

GO FROM <code_addr> TILL <code_addr> OR <code_addr>
This command acts like the previous command except that it also sets the second breakpoint (BP[1]) to the <code_addr> which follows the OR keyword.

GO TILL <code_addr>
This command sets the first breakpoint (BP[0]) to <code_addr> and then starts the execution of user code using the current setting of the user's PC and the breakpoint array.

GO TILL <code_addr> OR <code_addr>
This command acts like the previous command except that it also sets the second breakpoint (BP[1]) to the <code_addr> which follows the OR keyword.

HALT
This command stops execution of user code by forcing the processor to execute a jump to self instruction in a reserved location.

## Program Stepping

These commands allow stepping through programs one instruction at a time. Between instructions the iECM-96 commands can be used to check the state of the variables changed by the instruction to ensure that the program is operating properly. Stepping through code allows a far more detailed look at what is going on in the program. The price that is paid for this detail is that stepping does not occur in real time; this makes it difficult or perhaps impossible to use on code that is tied to real time events.

Stepping while interrupts are enabled would be confusing since interrupt service routines will be stepped through as well as sequential code. iECM-96 avoids this problem by artificially locking out interrupts while stepping, ignoring the state of the interrupt enable (EI) or interrupt mask.

Super-Stepping is similar to stepping except that interrupts are not artificially suppressed. Also, an interrupt service routine or a subroutine call (and the body of the subroutine that is called) is treated as one indivisible instruction by the super-step command. This allows the user to ignore the details of subroutines and interrupt service routines while checking out code. Every time an instruction is "super-stepped" all the service routines associated with enabled pending interrupts will be executed. This may allow limited stepping through code while operating in a concurrent environment but the system will not operate in real time. A better approach is to use the GO command to execute to a specified breakpoint and then step through the code being tested looking for proper operation.

iECM-96 implements the step operation by using the TRAP instruction. To step over a given instruction iECM-96 determines all the possible subsequent instructions and places TRAPS at these locations. After doing this it allows the user's program to execute until it runs into one of these TRAPS and then restores all of the user code bytes which were overwritten with TRAPS. If iECM-96 is to step over a conditional branch, two possible subsequent instructions exist in the sequential code of the program. Any other instruction can only have one "next" instruction. A TRAP is also set at location 2080H in case the target is reset during the step.

Super-stepping is accomplished by setting TRAPS like the STEP except for CALL instructions which are treated as a special case. During a STEP the iECM-96 will put the TRAP at the target address of a call; during a super-step the TRAP will be placed at the instruction following the CALL. Interrupts are suppressed during STEP (not SS) operations by saving the user's EI bit, clearing it before the STEP occurs, and then restoring it. In order to make sure the instruction which is executed does not modify the EI bit, several instructions (PUSHF, POPF, PUSHA, POPA, DI, EI) are simulated by the iECM-96 software rather than being executed by the target processor. The 80C196KB instruction IDLPD is also simulated during STEP to prevent the target from locking up. The simulation treats the IDLPD as a two byte NO-OP. Note that the simulation of instructions only occurs during STEP operations. During a GO or SS command all instructions are executed by the target.

The iECM-96 commands which implement step operations are:

STEP
STEP <count>
STEP FROM <code_addr>
STEP FROM <code_addr> <count>
SS
SS <count>
SS FROM <code_addr>
SS FROM <code_addr> <count>

Aside from the style of the actual step operation, the SS and STEP commands behave the same. They will be described together and will be called single-stepping.

{STEP | SS}
This command single-steps one time.

{STEP | SS } <count>
This command single-steps <count> times.

{ STEP | SS } FROM <code_addr>
This command loads the user's pc (PC) with <code_addr> and then single-steps one time.

{ STEP | SS } FROM <code_addr> <count>
This command loads the user's pc (PC) with <code_addr> and then single_steps <count> times.

## DISPLAYING AND MODIFYING PROGRAM VARIABLES

iECM-96 provides commands to display and modify program variables in several formats. In addition to simple variables such as bytes and words, more complicated variables such as reals and character strings are supported. iECM-96 commands allow variables to be displayed or initialized either individually or as regions of memory which contain variables of the given type.

### Supported Data Types

BYTE
A BYTE is an eight-bit variable. No alignment rules are enforced for BYTE variables.

CHAR
A CHAR is a special case of a BYTE. CHAR variables are displayed as ASCII characters.

WORD
A WORD is a 16-bit variable. The address of a WORD is the address of its least significant byte. A WORD must start at an even byte address.

DWORD
A DWORD is a 32-bit variable. The address of a DWORD is the address of its least significant byte. A DWORD must always start at an even byte address. If a DWORD variable is to be accessed as a register by an 8096 instruction then a more restrictive alignment rule is enforced: it must start at an address which is evenly divisible by 4. This more restrictive alignment rule will only apply to iECM-96 commands when using the single line assembler.

REAL
A REAL is a 32-bit binary floating point number which conforms to the FPAL96 definition. The 32 bits contain a sign bit, an 8-bit exponent field, and a 23-bit fraction field. iECM-96 commands use standard scientific notation to deal with REAL numbers. Note that the FPAL96 has special representations for ±infinity and for NaN's (Not a Number--used to signal error conditions) if iECM-96 detects one of these special values it will output an appropriate text string instead of trying to display the value in scientific notation.

STACK
A STACK variable is a 16-bit variable which resides in the system stack. The addresses of stack variables (<stack_addr> are taken to be relative to the current stack pointer and must be word aligned.

STRING
A STRING is a sequence of ASCII characters which are terminated by the NUL character. The ASCII character NUL has the binary value of zero.

In addition to supporting access to variables of the above types, iECM-96 also provides commands to access the special program variables PC (program counter), PSW (program status word) and SP (stack pointer). These commands are discussed at the end of this section under the heading "Processor Variables".

**BYTE Commands**

There are four forms for the BYTE commands:

BYTE <byte_address>
BYTE <byte_address> = <byte_value>
BYTE <byte_address> TO <byte_address>
BYTE <byte_address> TO <byte_address> = <byte_value>

All of these commands can be used whether or not the user's program is running.

BYTE <byte_address>
This form is used to examine and then possibly change one or more sequential BYTE variables. When this command is invoked iECM-96 will display the <byte_address> symbolically if a valid symbol exists for that <byte_address>. Whether or not the symbolic display occurs, iECM-96 will display the <byte_address> in hexadecimal notation, the value of the BYTE in the default base and wait for an input from you. You can respond with a CARRIAGE-RETURN character, an ESC character, or by entering a numeric value. A CARRIAGE-RETURN will terminate the command. An ESC will result in the display of the next sequential BYTE variable. If a numeric value is entered then the BYTE variable will be set to this value and the iECM-96 will again wait for input. At this point you can respond only with an ESC or CARRIAGE-RETURN. As before, the ESC will display the next sequential BYTE and the CARRIAGE-RETURN will terminate the command.

BYTE <byte_address> = <byte_value>
This form is used to set an individual BYTE variable without first checking its current value. When invoked, this command sets the BYTE variable at <byte_address> to <byte_value>.

BYTE <byte_address> TO <byte_address>
This form is used to display a region of memory as a sequence of BYTE variables. When this command is invoked, iECM-96 will start by displaying the current default base and then a series of lines showing the contents of the selected memory region. If a symbol exists in iECM-96's symbol table for the next <byte_address> then this symbol will be displayed. Whether or not the symbolic display happens, the next line will start with a hexadecimal display of the address of the next BYTE variable to be displayed followed by the display of up to 16 bytes of memory as BYTE variables in the default base. A new line will be started whenever 16 bytes of memory have been displayed on the line or a valid symbol exists in iECM-96's symbol table for the next <byte_address> to be displayed. The command terminates when all of the BYTE variables in the selected range have been displayed. During lengthy displays you can stop the output to the console by hitting the SPACE bar. Display can be resumed by hitting the SPACE bar a second time. The command can be terminated by entering a carriage return.

BYTE <byte_address> TO <byte_address> = <byte_value>This form is used to initialize a region of memory to the given <byte_value>. Note that this command will take a little over a millisecond (at 9600 baud) for each BYTE loaded. This command can be terminated by entering a carriage return but this leaves only part of the memory region initialized.

### WORD Commands

There are four basic forms for the WORD commands:

WORD <word_address>
WORD <word_address> = <word_value>
WORD <word_address> TO <word_address>
WORD <word_address> TO <word_address> = <word_value>

All of these commands can be used whether or not the user's program is running.

WORD <word_address>
This form is used to examine and then possibly change one or more sequential WORD variables. When this command is invoked iECM-96 will display the <word_address> symbolically if a valid symbol exists for that <word_address>. Whether or not the symbolic display occurs, iECM-96 will display the <word_address> in hexadecimal notation, the value of the WORD in the default base and wait for an input from you. You can respond with a CARRIAGE-RETURN character, an ESC character, or by entering a numeric value. A CARRIAGE-RETURN will terminate the command. An ESC will result in the display of the next sequential WORD variable. If a numeric value is entered then the WORD variable will be set to this value and the iECM-96 will again wait for input. At this point you can respond only with an ESC or CARRIAGE-RETURN. As before, the ESC will display the next sequential WORD and the CARRIAGE-RETURN will terminate the command.

WORD <word_address> = <word_value>
This form is used to set an individual WORD variable without first checking its current value. When invoked, this command sets the WORD variable at <word_address> to <word_value>.

WORD <word_address> TO <word_address
>This form is used to display a region of memory as a sequence of WORD variables. When this command is invoked, iECM-96 will start by displaying the current default base and then a series of lines showing the contents of the selected memory region. If a symbol exists in iECM-96's symbol table for the next <word_address> then this symbol will be displayed. Whether or not the symbolic display happens, the next line will start with a hexadecimal display of the address of the next WORD variable to be displayed followed by the display of up to 16 bytes of memory as WORD variables in the default base. A new line will be started whenever 16 bytes of memory have been displayed on the line or a valid symbol exists in iECM-96's symbol table for the next <word_address> to be displayed. The command terminates when all of the WORD variables in the selected range have been displayed. During lengthy displays you can stop the output to the console by hitting the SPACE bar. Display can be resumed by hitting the SPACE bar a second time. The command can be terminated by entering a carriage return.

WORD <word_address> TO <word_address> = <word_value>
This form is used to initialize a region of memory to the given <word_value>. Note that this command will take a little over a millisecond (at 9600 baud) for each WORD loaded. This command can be terminated by entering a carriage return but this leaves only part of the memory region initialized.

### DWORD Commands

There are four basic forms for the DWORD commands:

DWORD <dword_address>
DWORD <dword_address> = <dword_value>
DWORD <dword_address> TO <dword_address>
DWORD <dword_address> TO <dword_address> = <dword_value>

All of these commands can be used whether or not the user's program is running.

DWORD <dword_address>
This form is used to examine and then possibly change one or more sequential DWORD variables. When this command is invoked iECM-96 will display the <dword_address> symbolically if a valid symbol exists for that <dword_address>. Whether or not the symbolic display occurs, iECM-96 will display the <dword_address> in hexadecimal notation, the value of the DWORD in the default base and wait for an input from you. You can respond with a CARRIAGE-RETURN character, an ESC character, or by entering a numeric value. A CARRIAGE-RETURN will terminate the command. An ESC will result in the display of the next sequential DWORD variable. If a numeric value is entered then the DWORD variable will be set to this value and the iECM-96 will again wait for input. At this point you can respond only with an ESC or CARRIAGE-RETURN. As before, the ESC will display the next sequential DWORD and the CARRIAGE-RETURN will terminate the command.

DWORD <dword_address> = <dword_value>
This form is used to set an individual DWORD variable without first checking its current value. When invoked, this command sets the DWORD variable at <dword_address> to <dword_value>.

DWORD <dword_address> TO <dword_address>
This form is used to display a region of memory as a sequence of DWORD variables. When this command is invoked, iECM-96 will start by displaying the current default base and then a series of lines showing the contents of the selected memory region. If a symbol exists in iECM-96's symbol table for the next <dword_address> then this symbol will be displayed. Whether or not the symbolic display happens, the next line will start with a hexadecimal display of the address of the next DWORD variable to be displayed followed by the display of up to 16 bytes of memory as DWORD variables in the default base. A new line will be started whenever 16 bytes of memory have been displayed on the line or a valid symbol exists in iECM-96's symbol table for the next <dword_address> to be displayed. The command terminates when all of the DWORD variables in the selected range have been displayed. During lengthy displays you can stop the output to the console by hitting the SPACE bar. Display can be resumed by hitting the SPACE bar a second time. The command can be terminated by entering a carriage return.

DWORD <dword_address> TO <dword_address> = <dword_value>
This form is used to initialize a region of memory to the given <dword_value>. Note that this command will take a little over a millisecond (at 9600 baud) for each DWORD loaded. This command can be terminated by entering a carriage return but this leaves only part of the memory region initialized.

**REAL Commands**

There are four basic forms for the REAL commands:

REAL <real_address>
REAL <real_address> = <real_value>
REAL <real_address> TO <real_address>
REAL <real_address> TO <real_address> = <real_value>

All of these commands can be used whether or not the user's program is running.

REAL <real_address>
This form is used to examine and then possibly change one or more sequential REAL variables. When this command is invoked iECM-96 will display the <real_address> symbolically if a valid symbol exists for that <real_address>. Whether or not the symbolic display occurs, iECM-96 will display the <real_address> in hexadecimal notation, the value of the REAL in the default base and wait for an input from you. You can respond with a CARRIAGE-RETURN character, an ESC character, or by entering a numeric value. A CARRIAGE-RETURN will terminate the command. An ESC will result in the display of the next sequential REAL variable. If a numeric value is entered then the REAL variable will be set to this value and the iECM-96 will again wait for input. At this point you can respond only with an ESC or CARRIAGE-RETURN. As before, the ESC will display the next sequential REAL and the CARRIAGE-RETURN will terminate the command.

REAL <real_address> = <real_value>
This form is used to set an individual REAL variable without first checking its current value. When invoked, this command sets the REAL variable at <real_address> to <real_value>.

REAL <real_address> TO <real_address>This form is used to display a region of memory as a sequence of REAL variables. When this command is invoked, iECM-96 will display a series of lines showing the contents of the selected memory region. If a symbol exists in iECM-96's symbol table for the next <real_address> then this symbol will be displayed. Whether or not the symbolic display happens, the next line will start with a hexadecimal display of the address of the next REAL variable to be displayed followed by the display of up to 16 bytes of memory as REAL variables in the default base. A new line will be started whenever 16 bytes of memory have been displayed on the line or a valid symbol exists in iECM-96's symbol table for the next <real_address> to be displayed. The command terminates when all of the REAL variables in the selected range have been displayed. During lengthy displays you can stop the output to the console by hitting the SPACE bar. Display can be resumed by hitting the SPACE bar a second time. The command can be terminated by entering a carriage return.

REAL <real_address> TO <real_address> = <real_value>
This form is used to initialize a region of memory to the given <real_value>. Note that this command will take a little over a millisecond (at 9600 baud) for each REAL loaded. This command can be terminated by entering a carriage return but this leaves only part of the memory region initialized.

### STACK Commands

There are two basic forms for the STACK commands:

STACK <stack_address>
STACK <stack_address> TO <stack_address>

Both of these commands can be used whether or not the user's program is running.

STACK <stack_address>
This command is useful for accessing a 16-bit variable which is known to be a fixed offset in the system stack. When this command is invoked, iECM-96 executes a "WORD <word_address> command where the <word_addr> is formed by adding <stack_address> to the current value of the system stack pointer.

STACK <stack_address> TO <stack_address>
This command is useful for accessing a sequence of 16-bit variables which are known to start at a fixed offset in the system stack. When this command is invoked, iECM-96 executes a "WORD <word_address> TO <word_address> command where both <word_address> fields are formed by adding the corresponding <stack_address> to the current value of the system stack pointer. During lengthy displays you can stop the output to the console by hitting the SPACE bar. Display can be resumed by hitting the SPACE bar a second time. The command can be terminated by entering a carriage return.

### STRING commands

There is only one form of the STRING command:

STRING <byte_address>
If a symbol exists for <byte_address> in the iECM-96's symbol table then this symbol will be displayed. Whether or not the symbolic display happens, the next line will start with a hexadecimal display of <byte_address> followed by the NUL terminated ASCII string starting at that address. For long strings only the first 60 characters are displayed. When trailing characters are stripped, decimal points (".") are substituted for the first three characters stripped.

## Processor Variables

Several commands are provided to access variables which are associated with the processor rather than with the program:

PC
PC = <byte_address>
PSW
PSW = <word_value>
SP
SP = <word_address>

The processor variables can be modified only while the target is stopped, they can be read at any time. These commands allow the display and loading of the program counter (PC), program status word (PSW) and stack pointer (SP). Display is in the default base.

**NOTE: The examination of the SP will be confusing if you don't understand the following paragraphs.**

The iECM-96 software uses two words in the user's stack to store the PC and PSW during a host interface interrupt. When the user displays the SP (or uses the STACK command) the value shown for SP is adjusted by 4 bytes to compensate for this overhead so that it becomes more or less invisible to the user (the user must still allow for the extra stack space used). This is convenient but creates confusion if you display using the SP command and then use the WORD command to look at location 18H which is the register address of the stack pointer. Location 18H will be 4 less than "SP".

An additional consideration is what happens when you attempt to write into the stack pointer using the SP command. Before returning from the RISM interrupt service routine (ISR) which actually updates the stackpointer, the RISM places in the stack a return address and associated PSW for the idle loop it executes while the target is "stopped". This prevents the target from getting lost upon return from the ISR. You should not attempt to modify the stack pointer from the console through the use of its register address (18H); it should only be modified by the SP commands or by execution of user code in the target. This decreases the possibility of the target getting confused.

Specific implementations of the RISM may actually prevent the user from writing into "WORD 18" and thereby force the user to use the "SP" command.

## ASSEMBLY AND DISASSEMBLY

iECM-96 supports the examination and modification of code memory using the standard mnemonics for the MCS-96 assembler (ASM-96). Although standard mnemonics are used, the iECM-96 does not build a symbol table of user symbols as assembly mnemonics are entered. This makes it a single-line assembler (SLA) because references are never made to information entered on other lines. No labels are generated by the SLA, although it can use labels which are loaded as symbolic information along with object code when a file translated in the debug mode has been loaded. The iECM-96 SLA will accept mnemonics for all instructions which can actually be executed by the target processor. It will not accept "generic" instructions such as BE or CALL which are processed by ASM-96 into standard MCS-96 instructions. It will accept JE and SCALL or LCALL which are the specific instructions the MCS-96 processors understand.

### SLA (Single Line Assembly) Commands

The commands which invoke the SLA are:

ASM <code_address>
ASM

The SLA is useful for writing short code pieces on-line for testing or patching programs but is not intended as a replacement for a true assembler such as ASM-96. The SLA can be invoked whether or not user code is running, but there is an obvious danger in modifying code that is being executed.

ASM <code_addr>
This command causes the iECM-96 software to enter the SLA mode. The assembly program counter (APC) will be set to <code_addr> and lines of "assembly language" entered by the user will be converted to object code and loaded into the target's memory. iECM-96 will complain if erroneous inputs are made but will remain in the SLA mode. This mode is terminated by entering the only "directive" understood by the SLA: END.

ASM
This command operates identically to the ASM <code_addr> command except that the APC is not initialized. If this is the first time that the SLA has been used then APC will be set to 2080H, if it is not then APC will point at the byte following the last instruction generated by the SLA.

## Disassembly Commands

The disassembler converts binary object code in the target memory to ASM-96 mnemonics. There are several commands which invoke the disassembler:

DASM
DASM <count>
DASM <code_addr>
DASM <code_addr>,<count>
DASM <code_addr> TO <code_addr>

These commands are useful for examining a portion of the program for which listings are not available or for checking program patches, and can be used whether or not user code is running.

DASM
This command disassembles the instruction currently pointed to by the user's program counter (PC).

DASM <count>
This command reads the current value of the user's program counter (PC) and disassembles <count> instructions starting at that location. The parameter <count> must be less than 256T (100H) so that the command parser can distinguish this command from the command "DASM <code_addr>. This restriction does not apply to the DASM <code_addr>,<count> instruction. During lengthy displays you can stop the output to the console by hitting the SPACE bar. Display can be resumed by hitting the SPACE bar a second time. The command can be terminated by entering a carriage return.

DASM <code_addr>
This command disassembles the instruction at <code_addr>. The parameter <code_addr> must be greater or equal to 256T (100H) so that the command parser can distinguish it from the DASM <count> instruction.

DASM <code_addr>,<count>
This command disassembles <count> instructions starting with the one at <code_addr>. During lengthy displays you can stop the output to the console by hitting the SPACE bar. Display can be resumed by hitting the SPACE bar a second time. The command can be terminated by entering a carriage return.

DASM <code_addr> TO <code_addr>
This command disassembles the region of memory specified. If an instruction crosses the ending address of the region it will be completely disassembled before the command terminates. During lengthy displays you can stop the output to the console by hitting the SPACE bar. Display can be resumed by hitting the SPACE bar a second time. The command can be terminated by entering a carriage return.

## SYMBOL OPERATIONS

iECM-96 supports several commands dealing with symbolic information that can be loaded along with object code. The commands are:

SYMBOLS
SYMBOLS OFF
SYMBOLS ON
FLUSH

An additional command, "LOADSYM <filename>" can be used to load iECM-96's symbol table without affecting the target's memory. This command is described in the section "File Operations".

SYMBOLS
This command displays the symbols that are currently in iECM-96's symbol table.

SYMBOLS OFF
This command suppresses searching the symbol table during output. It does not prevent the use of the symbol table during input. This command is provided because symbolic output with large symbol tables can be very slow.

SYMBOLS ON
This command reenables symbolic output.

FLUSH
This command deletes all the symbols currently in the symbol table.

## RISM

This section will describe the elements of the RISM which will be common to all implementations. Additional documentation of this implementation is in appendices B and C.

### RISM Variables

RISM_DATA
RISM_DATA is a 32-bit register which acts as the primary data interface between software running in the host and the RISM running in the target.

RISM_ADDR
RISM_ADDR is a 16-bit register which contains the address to be used for reading and writing target memory.

RISM_STAT
RISM_STAT is an 8-bit register used to store RISM status and state information. This register contains the following Boolean flags:

> DLE_FLAG
> This flag indicates the next character received by the RISM should be treated as a data byte even if its value corresponds to an implemented command.
>
> RUN_FLAG
> This flag indicates that the target is running user code.
>
> TRAP_FLAG
> This flag indicates that the target was running user code but that a software TRAP occurred which suspended its execution.
>
> DIAGNOSTIC_FLAG
> This is an optional flag that indicates that the target is operating in a diagnostic mode. The details of this are implementation dependent.

USER_PC
USER_PC is used to save the user's program counter while the user's code is not executing.

USER_PSW
USER_PSW is used to save the user's program status word while the user's code is not executing.

Other Variables
Specific implementations of RISMs will require other variables to be used for temporary storage.

### RISM Structure

The RISM resides in the target system and provides the interface between the target system and the user interface which resides in the host system. A design goal of the RISM was to keep it compact and simple. This serves two purposes:

1. The RISM can reside in a user's system with minimal impact on available memory

2. The RISM is easy to port into the target's environment.

The goals were met by keeping the internal state structure of the RISM as simple as possible. There are only three internal flags which can change the way that the RISM deals with a character sent by the host.

DLE_FLAG: If this flag is set then the next received character is assumed to be a data byte as opposed to a command byte.

RUN_FLAG: This flag is set if the target is running user code. It can modify the operation of some of the RISM commands.

TRAP_FLAG: This flag is set if the user code has been halted because it executed a TRAP instruction. The TRAP_FLAG is cleared whenever the RISM starts the execution of user code.

### Receiving Data from the Host

When the RISM receives a character from the host its first task is to determine if it represents a command or data. If the character is less than 32 (decimal) then it is assumed to be a command, if not then it is taken to be data. If the host needs to send a data byte which has a value less than 32 then it first must issue a SET_DLE command. If the DLE_FLAG is set then the next character received by the RISM will be interpreted as data (even if it is less than 32) and then the DLE_FLAG will be cleared. Once the RISM has determined that the received character is a data byte it processes it by shifting the 32-bit RISM_DATA register left eight places and then placing the data byte in the lower byte of the RISM_DATA register. The data shifted out of the upper byte of the RISM_DATA register is discarded.

### Sending Data to the Host

When the host expects data to be returned from the RISM it sends a TRANSMIT command byte and waits for a response. The RISM transmits the lower byte of the 32-bit RISM_DATA register and right shifts the RISM_DATA register right by eight bits. As part of this command the RISM increments its RISM_ADDR register. The RISM only transmits data in response to an TRANSMIT command, never on its own initiative or even in response to other commands from the host.

## RISM Commands

This section will detail the operation of each of the commands sent to the RISM.

SET_DLE_FLAG ( Code 00H )
This command sets the DLE_FLAG. This will force the next character received by the RISM to be treated as data even if its value corresponds to a RISM command. The code which overrides the normal selection of command or data also clears the DLE_FLAG so that it applies only to the first character received after the SET_DLE_FLAG command.

TRANSMIT ( Code 02H )
This command will transmit the lower eight bits of the RISM_DATA register to the host, right shift the data register eight places, and increment the RISM_ADDR register. Sequential TRANSMIT commands are used to read the RISM_DATA register and the RISM_ADDR register indicates the address that corresponds to the least significant byte in the RISM_DATA register.

READ_BYTE ( Code 04H )
This command will read the byte of memory pointed to by the RISM_ADDR register and place the result in the least significant byte of the RISM_DATA register.

READ_WORD ( Code 05H )
This command will read the word of memory pointed to by the RISM_ADDR register and place the result in the least significant word of the RISM_DATA register.

READ_DOUBLE ( Code 06H )
This command will read the double-word of memory pointed to by the address register and place the result in the RISM_DATA register.

WRITE_BYTE ( Code 07H )
This command stores the least significant byte of the RISM_DATA register in the byte of memory pointed to by the RISM_ADDR register and increments the RISM_ADDR register (by one) to point at the next memory byte.

WRITE_WORD ( Code 08H )
This command stores the least significant word of the RISM_DATA register in the word of memory pointed to by the RISM_ADDR register and increments the RISM_ADDR register (by two) to point at the next memory word.

WRITE_DOUBLE ( Code 09H )
This command stores the RISM_DATA register in the double-word of memory pointed to by the RISM_ADDR register and increments the RISM_ADDR register (by four) to point at the next memory double-word.

LOAD_ADDRESS ( Code 0AH )
This command loads the RISM_ADDR register with the least significant word in the RISM_DATA register.

INDIRECT_ADDRESS ( Code 0BH )
This command reads the memory word pointed to by the RISM_ADDR and stores it into the RISM_ADDR register. The RISM_DATA register is not modified by this command.

READ_PSW ( Code 0CH )
This command loads the RISM_DATA register with the PSW (Program Status Word) associated with the user's code. Most RISM implementations will have to check RUN_FLAG to determine how to access the user's PSW.

WRITE_PSW (Code 0x0D)
This command loads the PSW (Program Status Word) associated with the user's code from the RISM_DATA register. The host software will only invoke this command while user code is not running.

READ_SP (Code 0x0E)
This command loads the RISM_DATA register with the SP (Stack Pointer) associated with the user's code.

WRITE_SP (Code 0x0F)
This command loads the SP (Stack Pointer) from the RISM_DATA register. This command must also push two values into the newly created stack area. These values are the PC (first) and PSW (second) associated with the idle loop which executes while user code is not running. The host software will only invoke this command while user code is not running.

READ_PC (Code 0x10)
This command loads the RISM_DATA register with the PC (Program Counter) associated with the user's code. Most RISM implementations will have to check RUN_FLAG to determine how to access the user's PC.

WRITE_PC (Code 0x11)
This command loads the PC (Program Counter) associated with the user's code from the RISM_DATA register. The host software will only invoke this command while user code is not running.

START_USER (Code 0x12)
This command is responsible for starting the execution of user code, clearing the TRAP_FLAG, and setting RUN_FLAG. The action of this command relies on it being executed as part of an ISR (interrupt service routine). At the start of the ISR the current PC and PSW are pushed into the stack. If the user code is not running the PC and PSW which are pushed into the stack will be associated with an idle loop which the RISM runs while it waits for an interrupt. The START_USER command deletes the PC and PSW from the stack and replaces them with USER_PC and USER_PSW. When control returns from the ISR the user's code will execute rather than the idle loop. The host software will not issue a GO command if the user code is already running.

STOP_USER (code 0x13)
This command is responsible for stopping the execution of user code and clearing the RUN_FLAG. The action of the HALT command mirrors that of the GO command. In the case of the HALT command the user's PC and PSW are pushed into the stack upon entry to the ISR. The STOP_USER command saves this user information in USER_PC and USER_PSW and replaces it with PC and PSW values which are associated with the idle loop. When control returns from the ISR the idle loop will execute rather than the user's code. The host software will not issue a HALT command unless the user code is running.

TRAP_ISR
This is a pseudo-command. It can not be issued directly by the host software but is executed when a TRAP instruction is executed. The TRAP instruction is used by iECM-96 to implement software breakpoints and single stepping. A separate entry point into the STOP_USER is provided for the TRAP vector. Code at this entry point sets the TRAP_FLAG and then drops into the code which implements the STOP_USER command.

REPORT_STATUS (Code 0x14)
This command loads the least significant word of the RISM_DATA register with status information. Valid status values are:

0--Indicates that user code is stopped
(RUN_FLAG and TRAP_FLAG are both FALSE).
1--Indicates that user code is running
(RUN_FLAG is TRUE)
2--Indicates that user code executed a TRAP instruction
(TRAP_FLAG is TRUE)

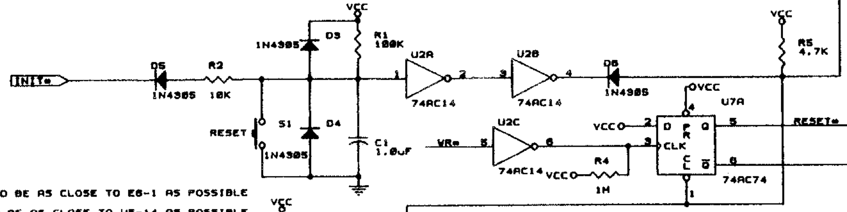
The host software will periodically poll the target system to check on its status and this polling can rob execution time from the user's program. This loss of target processor cycles can be avoided by setting the Ring Indicator modem status line signal whenever the RUN_FLAG is set. The host software will assume that the target is running user code whenever it detects the ring indicator and will only issue REPORT_STATUS commands if the ring indicator is off.

MONITOR_ESCAPE (Code 0x15)
This command provides for the addition of RISM commands for special purposes; it uses the RISM_DATA register to extend the command set of the RISM. The basic RISM requires only one of these "extended" commands; if the lower 16-bits of the RISM_DATA register is one (RISM_DATA = 0XXXX0001H) then the target processor should execute either a RST (ReSeT) instruction or a software initialization routine.

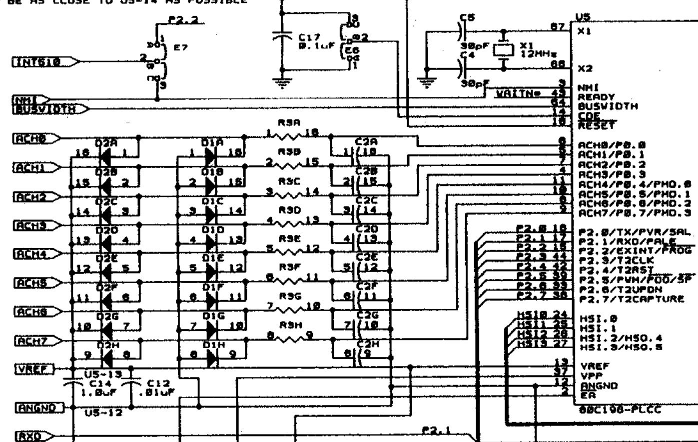
Start Up Commands ("/" or "\")
Upon reset the board is in the echo mode. Until it receives an ASCII slash ("/") or reverse-slash ("\") it should increment every character it receives from the host and send the incremented value back to the host. It will also display the binary code of the character received on the Port 1 LED's. If a reverse-slash is received by the RISM it will leave the echo mode (set USER_MAP flag true), remap memory and start normal operation. If a slash is received it will stop echoing incremented received data and start responding to RISM commands with the diagnostic flag set. In this mode there are diagnostic routine resident in EPROM which are useful for debugging the board. See the -DIAG option under Initiating and Terminating iECM-96 in the USER INTERFACE section of this manual for additional information on the Diagnostics Mode.

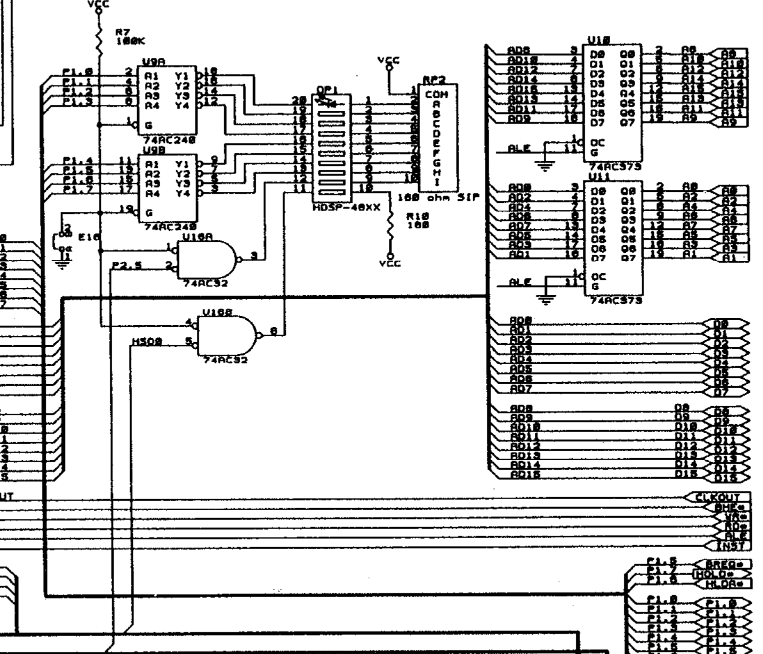
# Appendix A.

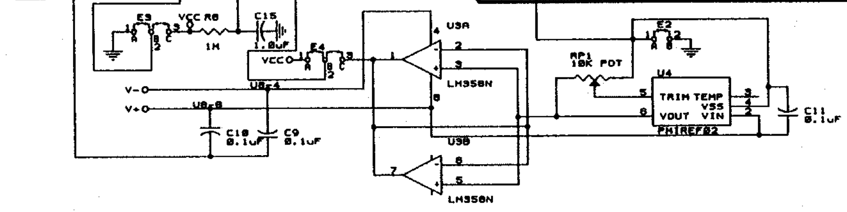
## Schematics and Parts List

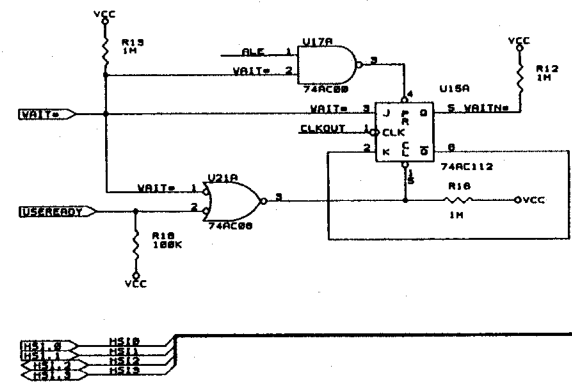


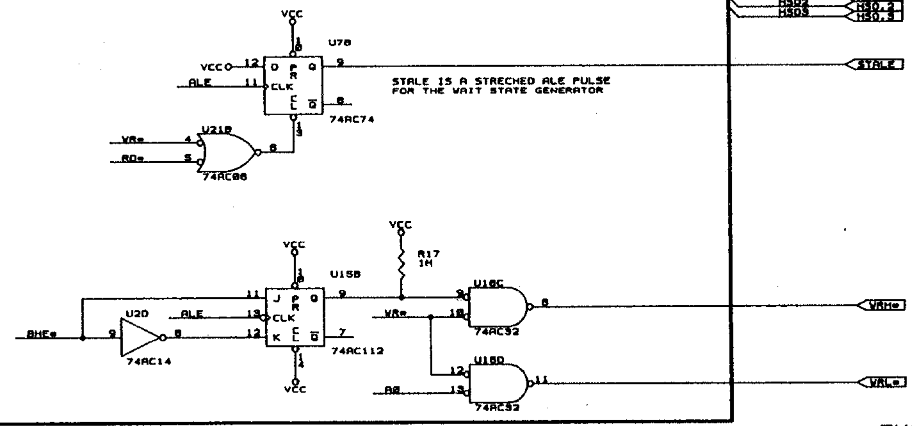
C17-2 SHOULD BE AS CLOSE TO E6-1 AS POSSIBLE
E6-2 SHOULD BE AS CLOSE TO U5-14 AS POSSIBLE





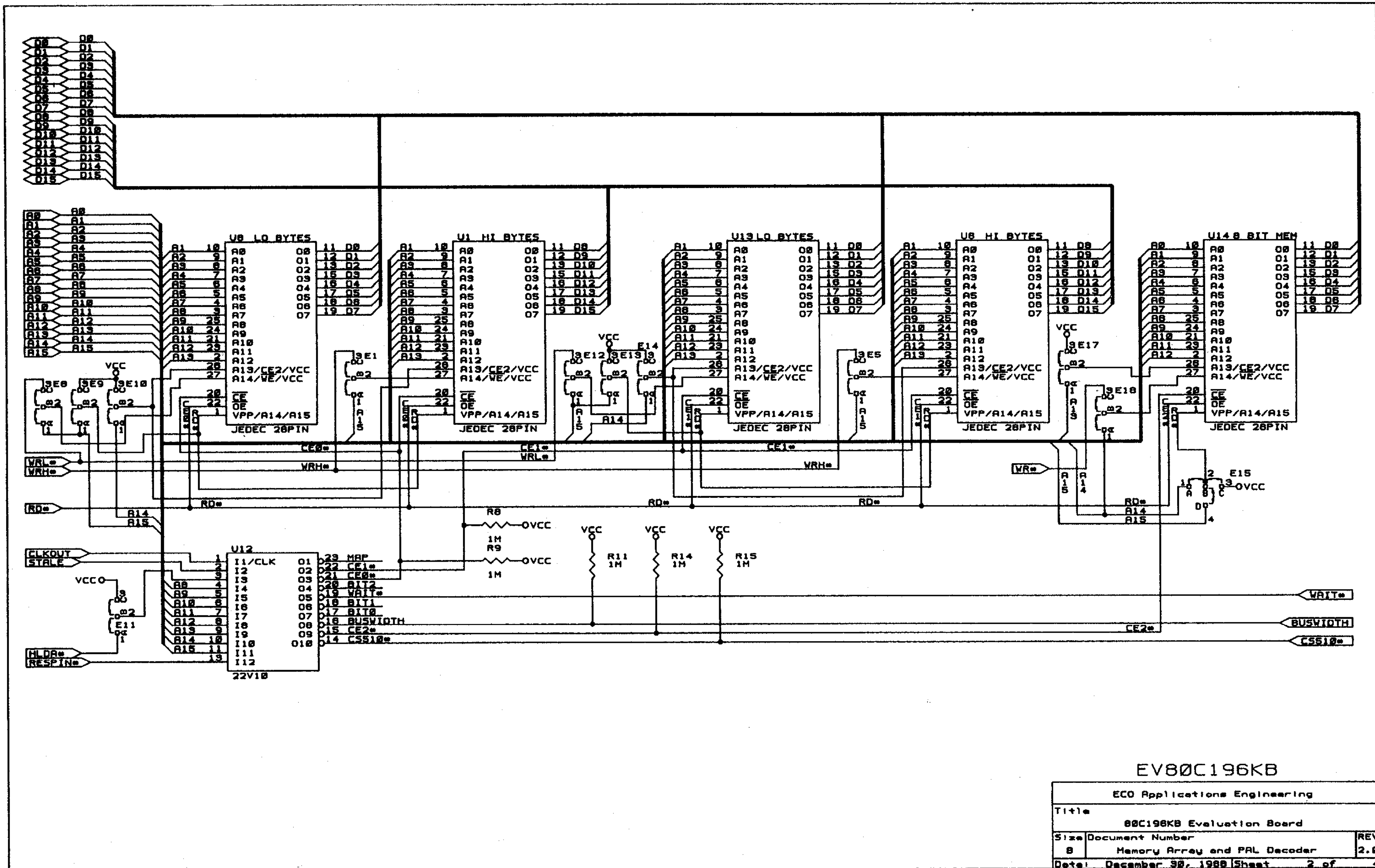




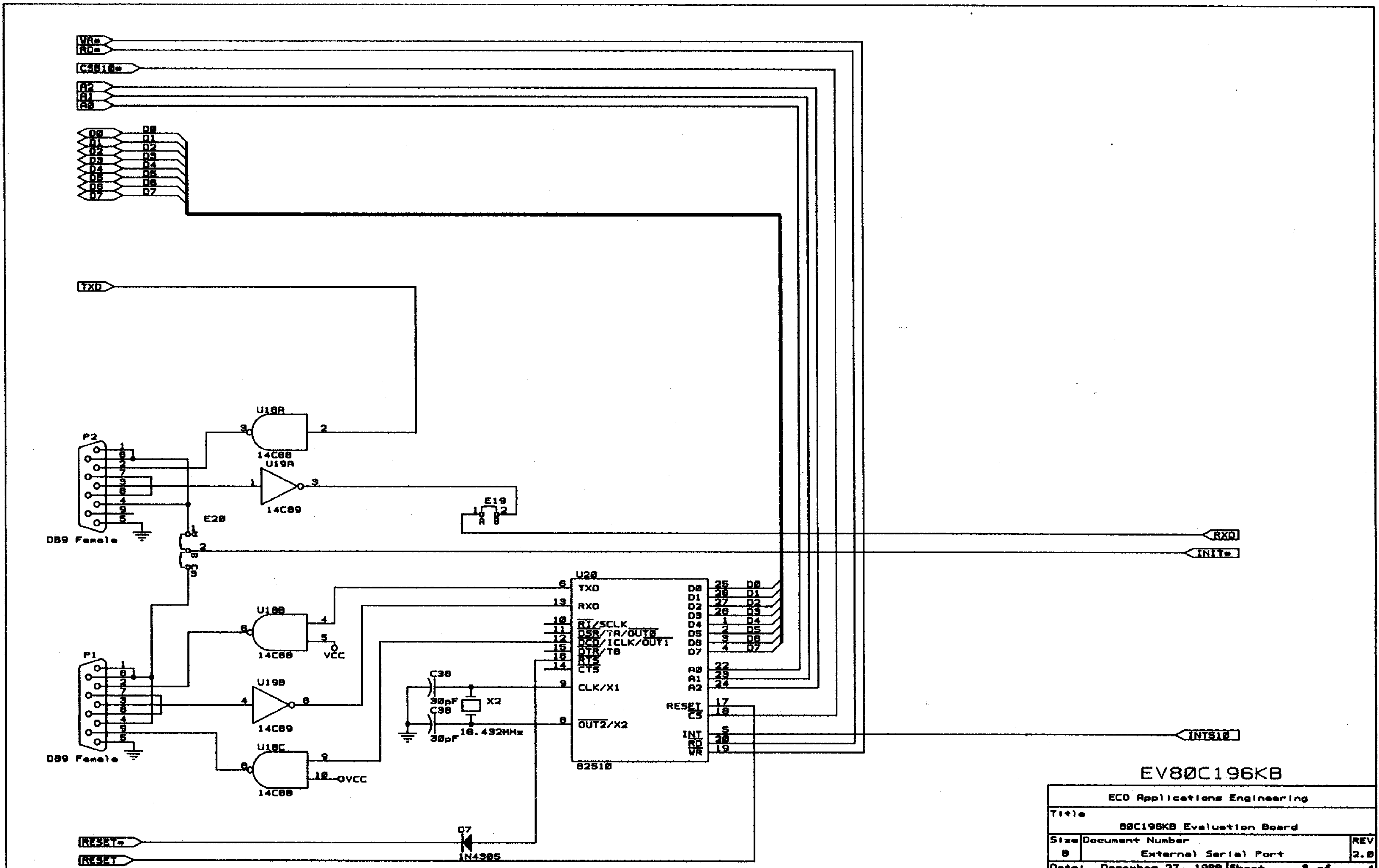



EV80C196KB

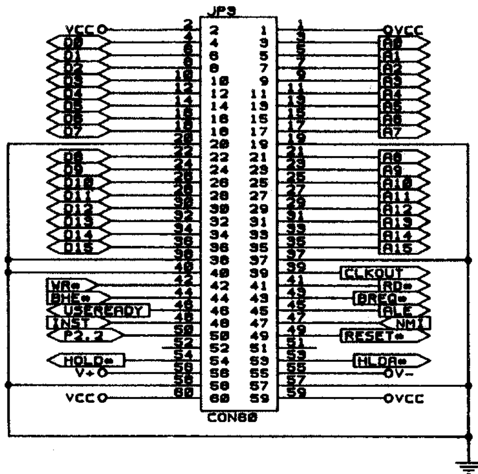
| ECO Applications Engineering | | |
|---|---|---|
| Title: 80C196KB Evaluation Board | | |
| Size: C | Document Number: CPU Section | REV: 2.0 |
| Date: January 10, 1989 | Sheet 1 of 4 | |

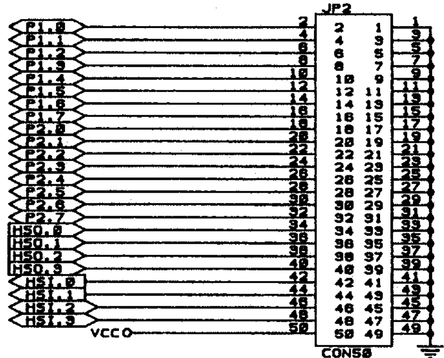


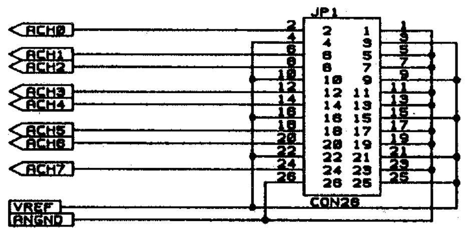
EV80C196KB
ECO Applications Engineering
Title
80C196KB Evaluation Board
Size B
Document Number
Memory Array and PAL Decoder
REV 2.0
Date: December 30, 1988 Sheet 2 of 4

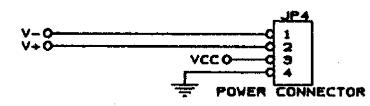


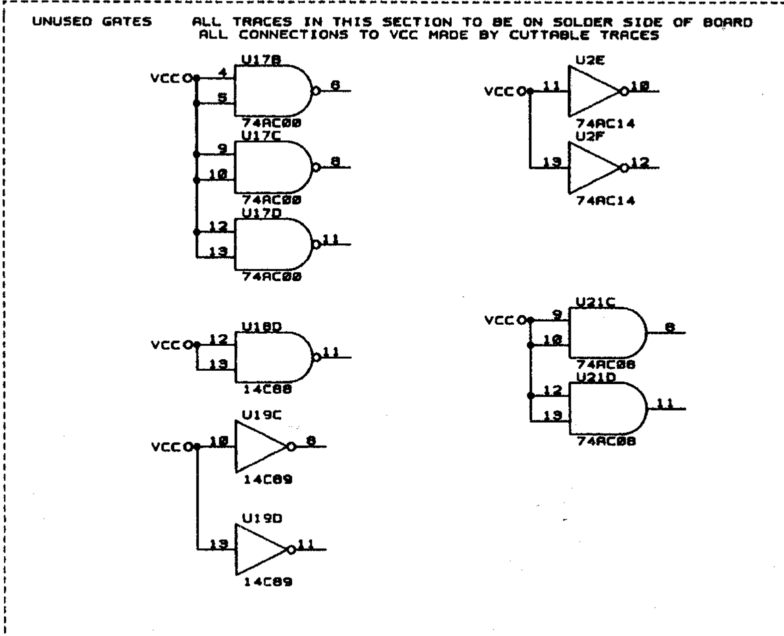

| ECO Applications Engineering | | |
|---|---|---|
| Title: 80C196KB Evaluation Board | | |
| Size: B | Document Number: External Serial Port | REV: 2.0 |
| Date: December 27, 1988 | Sheet 3 of 4 | |

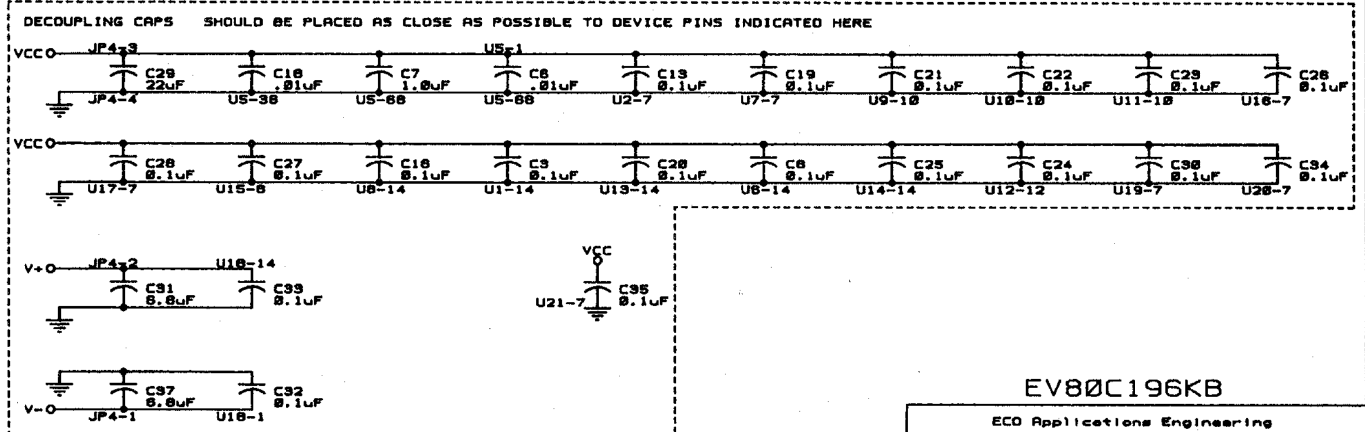













# EV80C196KB

| ECO Applications Engineering | | |
|---|---|---|
| Title | 80C196KB Evaluation Board | |
| Size | Document Number | REV |
| B | Exp. Busses, Spare Gates & Dcpl. Caps | 2.0 |
| Date: December 27, 1988 | Sheet 4 of 4 | |

80C196KB Evaluation Board
CPU Section

Revised: December 27, 1988
Revision: 2.0

ECO Applications Engineering

Bill Of Materials December 27, 1988 15:53:23 

| Item | Quantity | Reference | Part | Vendor | Manuf. | Part# |
|---|---|---|---|---|---|---|
| 1 | 1 | U5 | 80C196-PLCC | INTEL | INTEL | N80C196KB12 |
| 2 | 1 | U20 | 82510 | INTEL | INTEL | P82510 |
| 3 | 2 | U1,U8, | JEDEC 28PIN | INTEL | INTEL | D27C64 |
| 4 | 3 | U6,U13,U14 | JEDEC 28PIN | Sterling | Hitachi | HM6264P-10 |
| 5 | 1 | U17 | 74AC00 | Hamilton | Fairchild | 74AC00PC |
| 6 | 1 | U21 | 74AC08 | Hamilton | Fairchild | 74AC08PC |
| 7 | 1 | U2 | 74AC14 | Hamilton | Fairchild | 74AC14PC |
| 8 | 1 | U16 | 74AC32 | Hamilton | Fairchild | 74AC32PC |
| 9 | 1 | U7 | 74AC74 | Hamilton | Fairchild | 74AC74PC |
| 10 | 1 | U15 | 74AC112 | Hamilton | GE/RCA | CD74AC112E |
| 11 | 1 | U9 | 74AC240 | Hamilton | Fairchild | 74AC240PC |
| 12 | 2 | U10,U11 | 74AC373 | Hamilton | Fairchild | 74AC373PC |
| 13 | 1 | U18 | 14C88 | Hamilton | National | DS14C88N |
| 14 | 1 | U19 | 14C89 | Hamilton | National | DS14C89N |
| 15 | 1 | U12 | 22V10 | Luscombe | Cypress | PALC22V10-35PC |
| 16 | 1 | U4 | PMIREF02 | Hamilton | PMI | REF02HP |
| 17 | 1 | U3 | LM358N | Hamilton | | |
| 18 | 2 | D2,D1 | diode | Hamilton | | (8)1N4305 |
| 19 | 1 | R3 | resistor | Sterling | Dale | MDP-1603-271G |
| 20 | 1 | C2 | cap | Hamilton | Sprague | 926CX7R562K050B |
| 21 | 1 | X1 | 12MHz | Sterling | M-TRON | MP-1 12.0000 |
| 22 | 1 | X2 | 18.432MHz | Sterling | M-TRON | MP-1 18.4320 |
| 23 | 1 | S1 | RESET | Digi-key | Panisonic | P9950 |
| 24 | 5 | D5,D3,D4,D6,D7 | 1N4305 | Hamilton | | 1N4305 |
| 25 | 1 | DP1 | HDSP-48XX | Sterling | Lite-On | LTA1000G |
| 26 | 1 | R10 | 180 | Hamilton | Mepco | CR25-180 |
| 27 | 1 | R5 | 4.7K | Hamilton | Mepco | CR25-4.7K |
| 28 | 1 | R2 | 10K | Hamilton | Mepco | CR25-10K |
| 29 | 3 | R1,R7,R18 | 100K | Hamilton | Mepco | CR25-100K |
| 30 | 11 | R6,R4,R8,R9,R11,R12,R13, R14,R15,R16,R17 | 1M | Hamilton | Mepco | CR25-1M |
| 31 | 1 | RP2 | 180 ohm SIP | Hamilton | Bourns | 4610X-101-181 |
| 32 | 1 | RP1 | 10K POT | Hamilton | Bourns | 3009P-1-103 |
| 33 | 4 | C4,C5,C36,C38 | 30pF | Hamilton | Sprague | 1C10C0G330J050B |
| 34 | 3 | C12,C6,C18 | .01uF | Hamilton | Sprague | 1C10Z5U103M050B |
| 35 | 23 | C11,C3,C8,C9,C10,C13,C16, C17,C19,C20,C21,C22,C23, C24,C25,C26,C27,C28,C30, C32,C33,C34,C35 | 0.1uF | Hamilton | Sprague | 1C10Z5U104M050B |
| 36 | 4 | C1,C7,C14,C15 | 1.0uF | Hamilton | Sprague | 150D105X9015A2 |
| 37 | 2 | C31,C37 | 6.8uF | Hamilton | Sprague | 199D685X9035DA1 |

80C196KB Evaluation Board
CPU Section

Revised: December 27, 1988
Revision: 2.0

ECO Applications Engineering

Bill Of Materials December 27, 1988 15:53:23 

| Item | Quantity | Reference | Part | Vendor | Manuf. | Part# |
|---|---|---|---|---|---|---|
| 38 | 1 | C29 | 22uF | Hamilton | Sprague | 150D226X9015B2 |
| 39 | 2 | P1,P2 | DB9 Female | Sterling | AMP | 207084-1 |
| 40 | 3 | E2,E16,E19 | 2PIN JUMPER | Marshall | A P Prod. | |
| 41 | 16 | E7,E1,E3,E4,E5,E6,E8,E9, E10,E11,E12,E13,E14,E17, E18,E20 | 3PIN JUMPER | Marshall | A P Prod. | |
| 42 | 1 | E15 | 4PIN JUMPER | Marshall | A P Prod. | |
| 43 | 1 | JP4 | POWER CONNECTOR | Hamilton | Molex | 09-74-1041 |
| 44 | 1 | JP1 | CON26 | Marshall | A P Prod. | 929665-01-36 |
| 45 | 1 | JP2 | CON50 | Marshall | A P Prod. | 929665-01-36 |
| 46 | 1 | JP3 | CON60 | Marshall | A P Prod. | 929665-01-36 |

# Appendix B.

## Specific iRISM Information

## APPENDIX B

### Specific iRISM Information

The EV80C196KB is designed to be a software evaluation tool for the ROMless 80C196KB 16-bit microcontroller. As such, ports 3 and 4 are not available for use as I/O ports unless offboard latches/buffers and decoding logic are used. All unreserved functions of the 80C196KB are available to you except for the Non-Maskable Interrupt (NMI), the TRAP instruction, and 512 bytes of address space. The Chip Configuration Byte is also used by the monitor, but most of its functions are provided by external logic.

### Reserved Functions

The NMI pin is reserved for use by the Host Interface. In order for the Host Interface to function properly, jumper-shunt E7 must be installed from B-C. However, if your application demands the use of NMI (available on JP3), you can alter the RISM source file (96KBRISM.A96, included on your disk) to use EXTINT instead of NMI, and change jumper-shunt E7 to A-B.

The TRAP instruction is reserved.

On the EV80C196KB jumper shunt E20 must be installed from B to C for the RESET SYSTEM command to work properly. If you wish to run code in the board while it is not connected to a host, you should remove jumper shunt E20 prior to disconnecting the board from the host. If E20 is left installed, the board may reset as the connection is broken.

### Reserved Memory

User ROMsim as shipped is 24K bytes from address 2000H to 7FFFH. The board is reconfigurable to accept various memory devices. However, breakpoints and program stepping will not operate when your code is in EPROM or other nonchangeable memory. Normally you should write your code to begin at address 2080H and download it to ROMsim using iECM-96.

Two words of user stack space must be reserved for use by the iRISM-96 software while the board is processing a host interrupt.Register locations 30H-38H are reserved for use by the iRISM monitor code. You must ensure that no registers in this partition are used by code which is to operate with the RISM. The easiest way of doing this is to generate an ASM-96 module which declares an RSEG at 30H which is nine bytes long. This module can then be linked into the final program to prevent the linker from assigning these registers to some other module.

You must not alter the TRAP vector at 2010H or the NMI vector at 203EH.

Memory from 2014H-202FH is reserved for use by the iRISM monitor.

# Appendix C.

## Listing of iRISM-196KB

DOS 3.20 (038-N) MCS-96 MACRO ASSEMBLER, V1.2

SOURCE FILE: 96KBRISM.A96
OBJECT FILE: 96KBRISM.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: DEBUG

```
ERR LOC  OBJECT                 LINE       SOURCE STATEMENT
                                   1          EV96 module main
                                   2    ;        ================
                                   3    ;
                                   4    ; This file contains a RISM designed to operate the EV80C196KB evaluation
                                   5    ; board.  It includes the required RISM features and the optional diagnostic
                                   6    ; mode.  The board also supports remapping the memory space after reset.
                                   7    ; This allows the RISM code to gain control on reset and, after the
                                   8    ; initialization routines are complete, remap memory so that user code
                                   9    ; can be loaded into RAM at the reset location (2080H).
                                  10    ;
                                  11    ; The serial link is provided by an external UART (82510) with the received
                                  12    ; data interrupt tied to the NMI (Non Maskable Interrupt) of the processor.
                                  13    ; The use of the NMI for this purpose allows the user to maintain control
                                  14    ; of the system even if the running program locks out the interrupts or
                                  15    ; modifies the mask register.
                                  16    ;
                                  17    ; In addition to the NMI and its vector, this RISM uses the following
                                  18    ; resources:
                                  19    ;
                                  20    ;       Two words in the system stack
                                  21    ;
                                  22    ;       The TRAP instruction and its vector
                                  23    ;
                                  24    ;       External memory partitions (0000H-00FFH),
                                  25    ;                                  (1D00H-1EFFH), and
                                  26    ;                                  (2014H-202FH)
                                  27    ;
                                  28    ;               ( Note that all of these partitions, (except 1D00H-1EFFH and
                                  29    ;                 2018H), are reserved by the MCS-96 architecture. )
                                  30    ;
                                  31    ;       Nine bytes of registers in the partition  (30H-38H).  The
                                  32    ;       user must ensure that no registers in this partition are used
                                  33    ;       by code which is to operate with the RISM.  The easiest way of
                                  34    ;       doing this is to generate an ASM-96(tm) module which declares an
                                  35    ;       RSEG at 30H which is nine bytes long.  This module can then be
                                  36    ;       linked into the final program to prevent the linker from assigning
                                  37    ;       these registers to some other module.
                                  38    ;
                                  39    ;
                                  40    $eject
```

```
ERR LOC  OBJECT            LINE        SOURCE STATEMENT
                             41   ;
                             42   ; Define symbols for the register mapped I/O locations
                             43   ; ------------------------------------------------
                             44   ;
    0000                     45   zero            equ   00H:word      ; R/W  Zero Register
    0002                     46   ad_command      equ   02H:byte      ;   W  A to D command register
    0002                     47   ad_result_lo    equ   02H:byte      ; R    Low byte of result and channel
    0003                     48   ad_result_hi    equ   03H:byte      ; R    High byte of result
    0003                     49   hsi_mode        equ   03H:byte      ;   W  Controls HSI transition detector
    0004                     50   hsi_time        equ   04H:word      ; R    HSI time tag
    0004                     51   hso_time        equ   04H:word      ;   W  HSO time tag
    0006                     52   hsi_status      equ   06H:byte      ; R    HSI status register (reads fifo)
    0006                     53   hso_command     equ   06H:byte      ;   W  HSO command tag
    0007                     54   sbuf            equ   07H:byte      ; R/W  Serial port buffer
    0008                     55   int_mask        equ   08H:byte      ; R/W  Interrupt mask register
    0009                     56   int_pending     equ   09H:byte      ; R/W  Interrupt pending register
    0011                     57   spcon           equ   11H:byte      ;   W  Serial port control register
    0011                     58   spstat          equ   11H:byte      ; R    Serial port status register
    000A                     59   watchdog        equ   0AH:byte      ;   W  Watchdog timer
    000A                     60   timer1          equ   0AH:word      ; R    Timer1 register
    000C                     61   timer2          equ   0CH:word      ; R    Timer2 register
    000E                     62   port0           equ   0EH:byte      ; R    I/O port 0
    000E                     63   baud_reg        equ   0EH:byte      ;   W  Baud rate register
    000F                     64   ioport1         equ   0FH:byte      ; R/W  I/O port 1
    0010                     65   ioport2         equ   10H:byte      ; R/W  I/O port 2
    0015                     66   ioc0            equ   15H:byte      ;   W  I/O control register 0 (HSI/O)
    0015                     67   ios0            equ   15H:byte      ; R    I/O status register 0
    0016                     68   ioc1            equ   16H:byte      ;   W  I/O control register 1 (Port2)
    0016                     69   ios1            equ   16H:byte      ; R    I/O status register 1
    0017                     70   pwm_control     equ   17H:byte      ;   W  PWM control register
    0018                     71   sp              equ   18H:word      ; R/W  System stack pointer
                             72   ;
                             73   ; This section defines utility macros non-specific to this program
                             74   ; ----------------------------------------------------------------
                             75   ;
                             76   DEFINE_BIT      macro   name,bitnum
                             77                   name    equ bitnum
                             78                   endm
                             79
                             80   SET_BIT         macro   regnum,bitnum
                             81                   orb     regnum,#( 1 SHL (bitnum mod 8)  )
                             82                   endm
                             83
                             84   CLR_BIT         macro   regnum,bitnum
                             85                   andb    regnum,#not( 1 SHL (bitnum mod 8) )
                             86                   endm
                             87
                             88    BL             macro   label
                             89                   bnc     label
                             90                   endm
                             91   $eject
```

```
ERR LOC  OBJECT            LINE        SOURCE STATEMENT
                             92    ;
                             93    ; This section contains EQUates which may change with different versions
                             94    ; ----------------------------------------------------------------------
                             95    ;
      8000                   96    offset  equ     8000H           ; Code offset before REMAP
                             97    ;
                             98    ; Tell the commands what to use for psw while monitor is running
                             99    ;
      0000                  100    rism_psw        equ     0000H           ; No Interrupts enabled
                            101    ;
                            102    ; This section contains several macros generate specifically for this program
                            103    ; ---------------------------------------------------------------------------
                            104    ;
                            105    ; ENTER_RISM
                            106    ;       A macro which generates the prologue for the RISM ISR
                            107    ;
                            108    ; EXIT_RISM
                            109    ;       A macro which generates the epilogue for the RISM ISR
                            110    ;
                            111    ; SEND_DATA_BYTE
                            112    ;       A macro which passes the lower eight bits of RISM_DATA to
                            113    ;       the serial port, it assumes the port is ready for data
                            114    ;
                            115    ; BYTE_PROTECT
                            116    ;       A macro which terminates the RISM ISR if the RISM is about
                            117    ;       to write into a byte it should not modify.
                            118    ;
                            119    ; WORD_PROTECT
                            120    ;       A macro which terminates the RISM ISR if the RISM is about
                            121    ;       to write into a word it should not modify.
                            122    ;
                            123    ; DWORD_PROTECT
                            124    ;       A macro which terminates the RISM ISR if the RISM is about
                            125    ;       to write into a double-word it should not modify.
                            126    ;
                            127    $eject
```

```
ERR LOC  OBJECT          LINE       SOURCE STATEMENT
                          128   ENTER_RISM      macro
                          129                   pushf
                          130                   endm
                          131
                          132   EXIT_RISM       macro
                          133                   popf
                          134                   ret
                          135                   endm
                          136
                          137   SEND_DATA_BYTE  macro
                          138                   stb     RISM_DATA, txd_rxd[0]
                          139                   endm
                          140
                          141   BYTE_PROTECT    macro                   ; No special protection
                          142                   endm
                          143
                          144   WORD_PROTECT    macro                   ; No special protection
                          145                   endm
                          146
                          147   DWORD_PROTECT   macro                   ; No special protection
                          148                   endm
                          149   $eject
```

```
ERR LOC  OBJECT            LINE        SOURCE STATEMENT
                            150   ;
                            151   ; These registers are used only by the diagnostic routines.
                            152   ; ----------------------------------------------------------
                            153   ; They are not required for normal execution.
                            154   ;
    001C                    155           rseg at 1ch
                            156   ;       -----------
                            157   ;
    001C                    158   ax:             dsw     1
      001C                  159     al          equ  ax:byte
      001D                  160     ah          equ  (ax+1):byte
    001E                    161   dx:             dsw     1
    0020                    162   bx:             dsw     1
    0022                    163   cx:             dsw     1
                            164   ;
                            165   $eject
```

```
ERR LOC  OBJECT            LINE       SOURCE STATEMENT
                            166
                            167   ; These registers MUST be reserved for the RISM
                            168   ; --------------------------------------------
                            169   ;
    0030                    170           rseg at 30H
                            171   ;       -----------
                            172
    0030                    173   RISM_DATA:              dsl     1       ; The RISM data register
    0034                    174   RISM_ADDR:              dsw     1       ; The RISM address register
                            175   ;
    0036                    176   tempw:                  dsw     1       ; Temp for use by monitor
      0036                  177     tempb         equ     tempw:byte
      0036                  178     char          equ     tempw:byte
                            179   ;
    0038                    180   RISM_STAT:      dsb     1       ; Contains rism state flags
                            181           DEFINE_BIT      DLE_FLAG,0
                            183           DEFINE_BIT      RUN_FLAG,2
                            185           DEFINE_BIT      TRAP_FLAG,1
                            187           DEFINE_BIT      USER_MAP,3
                            189           DEFINE_BIT      DIAGNOSTIC_FLAG,7
                            191
                            192   ;
                            193   ; These variables are used by the monitor when in diagnotic mode only.
                            194   ; -------------------------------------------------------------------
                            195   ;
    003A                    196   dUSER_PC:               dsw     1       ; Saves user's pc during halt
    003C                    197   dUSER_PSW:              dsw     1       ; Saves user's psw during halt
                            198   ;
                            199   ;
    2020                    200           dseg at 2020H
                            201   ;       -------------
                            202   ; These variables are used in the normal (non-diagnostic) mode
                            203   ; ------------------------------------------------------------
                            204   ;
    2020                    205   USER_PC:                dsw     1       ; Saves user's pc during halt
    2022                    206   USER_PSW:               dsw     1       ; Saves user's psw during halt
                            207   $eject
```

```
ERR LOC  OBJECT         LINE       SOURCE STATEMENT
                        208
                        209     ; The serial channel is provided by an external 82510 UART which uses the NMI
                        210     ; as an interrupt to the processor.  The addresses associated with this
                        211     ; device are defined below.
                        212     ;
    1E00                213             dseg at 1E00H
                        214     ;       -------------
    1E00                215     uart:           dsb     100H
                        216     ;
      1E00              217             txd_rxd         equ     uart    :byte   ; bank0 (if dlab=0) or bank1
      1E00              218             baud_a_lo       equ     uart    :byte   ; bank0 (if dlab=1)
      1E01              219             baud_a_hi       equ     uart+1  :byte   ; bank0 (if dlab=1)
      1E01              220             gener_enabl     equ     uart+1  :byte   ; bank0 (if dlab=0)
      1E02              221             general_int     equ     uart+2  :byte   ; bank0
      1E03              222             line_config     equ     uart+3  :byte   ; bank0
      1E04              223             modem_contr     equ     uart+4  :byte   ; bank0
      1E05              224             line_status     equ     uart+5  :byte   ; bank0
      1E06              225             modem_stats     equ     uart+6  :byte   ; bank0
      1E07              226             addr_contr0     equ     uart+7  :byte   ; bank0
      1E00              227             clock_confg     equ     uart    :byte   ; bank3
      1E04              228             io_mode         equ     uart+4  :byte   ; bank3
                        229
                        230     ;
                        231     ; The memory map of the board is changed by reading or writing to an
                        232     ; address between 1000H and 1DFFH.  In this code, this is accomplished by
                        233     ; branching to address 1000H to continue RISM execution.  The memory map
                        234     ; of this board, both before and after RESET, are as follows:
                        235     ;
                        236     ; Address               After RESET             After REMAP
                        237     ;
                        238     ; 0000-00FFH as data    Internal Reg. file      Internal Reg. file
                        239     ; 0000-00FFH as code    RISM Monitor EPROM      RISM Monitor EPROM
                        240     ; 0100-1CFFH            Unused                  Unused--User expansion possible
                        241     ; 1D00-1DFFH            RISM Monitor EPROM      RISM Monitor EPROM
                        242     ; 1E00-1EFFH            External UART (U20)     External UART (U20)
                        243     ; 1F00-1FFFH            Unused (Port 3 & 4)     Unused (Port 3 & 4)
                        244     ; 2000-2013H            RISM Int. Vect. EPROM   User Int. Vect. RAM (NOT TRAP!)
                        245     ; 2014-202FH            RISM EPROM              RISM Data RAM
                        246     ; 2030-203FH            RISM Int. Vect. EPROM   User Int. Vect. RAM (NOT NMI!)
                        247     ; 2040-207FH            Unused RISM EPROM       User Data RAM
                        248     ; 2080-27FFH            RISM Monitor EPROM      User 16-Bit Code/Data RAM
                        249     ; 2800-5FFFH            16-Bit Code/Data RAM    User 16-Bit Code/Data RAM
                        250     ; 6000-7FFFH            8-Bit Code/Data RAM     User 8-Bit Code/Data RAM
                        251     ; 8000-FFFFH            Unused                  Unused--User expansion possible
                        252     ;
                        253     $eject
```

```
ERR LOC  OBJECT              LINE        SOURCE STATEMENT
                              254
    A000                      255           cseg at (offset + 2000H)
                              256    ;      ------------------------
                              257    ; Interrupt service routine addresses to be used in RISM EPROM.
                              258    ; Note:
                              259    ;   Of all these interrupt vectors, only the NMI and TRAP vectors are required
                              260    ;   for operation of the RISM.  The other vectors are provided as fixed entry
                              261    ;   points for routines which may be loaded into RAM in the diagnostic mode.
                              262    ;   In the diagnostic mode memory at the interrupt vectors is mapped to EPROM
                              263    ;   so it is not possible to write into the vector table.
                              264    ;
                              265    ;  (In the normal (i.e. non-diagnostic mode) the interrupt vector table is
                              266    ;  mapped to RAM so the vectors can be loaded as part of the normal process
                              267    ;  of loading a user's object code.
                              268    ;
    A000 0040                 269    timer_overflow:        dcw     4000H
    A002 0041                 270    ad_done:               dcw     4100H
    A004 0042                 271    hsi_data:              dcw     4200H
    A006 0043                 272    hso_event:             dcw     4300H
    A008 0044                 273    hsi_zero:              dcw     4400H
    A00A 0045                 274    software_timer:        dcw     4500H
    A00C 0046                 275    serial_port:           dcw     4600H
    A00E 0047                 276    external_int:          dcw     4700H
    A010 3B1D                 277    trap:                  dcw     (break-offset)
    A012 0048                 278    invalid_opcode:        dcw     4800H
                              279    ;
    A018                      280           cseg at (offset + 2018H)
                              281    ;      ------------------------
                              282    ;
    A018 FF                   283    chip_config:           dcb     0FFH    ; Enable no CCB modes
                              284    ;
    A030                      285           cseg at (offset + 2030H)
                              286    ;      ------------------------
                              287    ;
    A030 0049                 288    serial_txd:            dcw     4900H
    A032 004A                 289    serial_rxd:            dcw     4A00H
    A034 004B                 290    hsi_entry_4:           dcw     4B00H
    A036 004C                 291    timer2_capture:        dcw     4C00H
    A038 004D                 292    timer2_overflow:       dcw     4D00H
    A03A 004E                 293    external_int_pin:      dcw     4E00H
    A03C 004F                 294    hsi_fifo_full:         dcw     4F00H
    A03E 0000                 295    nmi:                   dcw     (rism_isr-offset)
                              296    ;
                              297    $eject
```

```
ERR LOC  OBJECT            LINE        SOURCE STATEMENT
                            298
    A080                    299         cseg at (offset + 2080H)
                            300   ;     ------------------------
                            301   ;
    A080                    302   reset_vector:
    A080 FA                 303         di
    A081 A1000118           304         ld      sp,#100H             ; Initialize stack pointer
    A085 3516FD             305         bbc     ios1,5, $            ; wait for a timer1 overflow
    A088 3516FD             306         bbc     ios1,5, $            ; ...two times,
    A08B C301002000         307         st      zero, 2000H          ; release uart reset, and wait
    A090 3516FD             308         bbc     ios1,5, $            ; ...till uart is ready
    A093 1138               309         clrb    RISM_STAT            ; Initialize rism mode register
                            310   ;
    A095 B18036             311         ldb     tempb, #80H          ; set dlab bit in line_config reg...
    A098 C701031E36         312         stb     tempb, line_config[0]  ; so that baud_a reg's are accessable
                            313   ;
    A09D B13C36             314         ldb     tempb, #3CH          ; set baud rate to 9600
    A0A0 C701001E36         315         stb     tempb, baud_a_lo[0]
    A0A5 C701011E00         316         stb     zero, baud_a_hi[0]
                            317   ;
    A0AA B10336             318         ldb     tempb, #03H          ; set up uart line config reg for no...
    A0AD C701031E36         319         stb     tempb, line_config[0]  ; par, 1 stop, 8bit, and txd_rxd access
                            320   ;
    A0B2 B16036             321         ldb     tempb, #60H          ; switch to bank3
    A0B5 C701021E36         322         stb     tempb, general_int[0]
                            323   ;
    A0BA B15036             324         ldb     tempb, #50H          ; select baud rate gen. a for both...
    A0BD C701001E36         325         stb     tempb, clock_confg[0]  ; rx and tx clock source
                            326   ;
    A0C2 B17F36             327         ldb     tempb, #7FH          ; select OUT1 mode on pin 12
    A0C5 C701041E36         328         stb     tempb, io_mode[0]
                            329   ;
    A0CA C701021E00         330         stb     zero, general_int[0]   ; switch to bank0
                            331   ;
    A0CF B10136             332         ldb     tempb, #01H          ; enable recieve fifo interrupt...
    A0D2 C701011E36         333         stb     tempb, gener_enabl[0]  ; of the uart
                            334   ;
    A0D7 A1000036           335         ld      tempw, #rism_psw     ; value for rism and initial user value
    A0DB C836               336         push    tempw                ; Set up psw for the monitor
    A0DD F3                 337         popf                         ; load psw with rism value
                            338   ;
    A0DE 1136               339         clrb    char
    A0E0 28F1               340         call    flash_leds           ; show life to user
    A0E2 27FE               341         br      $                    ; wait for interrupt
                            342   ;
                            343   $eject
```

```
ERR LOC  OBJECT            LINE      SOURCE STATEMENT
                            344
                            345   ; This code is entered from the nmi_isr if the user memory map is not turned
                            346   ; on.  This is the echo mode and diagnostic mode of the board.
                            347   ;
                            348   ; If the diagnostic flag is clear, the board is in echo mode.  Any characters
                            349   ; received from the host are incremented and sent back to the host.  They
                            350   ; are also tested for the set user command ('\') or the set diagnostics
                            351   ; command ('/').  If either command was sent it is carried out.
                            352   ;
                            353   ; If the diagnostic flag is set, the program branches to the diag. mode code.
                            354   ; ---------------------------------------------------------------------------
                            355   ;
    A0E4                    356   not_user:
    A0E4 3F3849             357           bbs     RISM_STAT, DIAGNOSTIC_FLAG, diag_mode
    A0E7 C40F36             358           stb     char, ioport1           ; splash received char on leds
    A0EA 1736               359           incb    char                    ; send back incremented char
    A0EC C701001E36         360           stb     char, txd_rxd[0]
    A0F1 1536               361           decb    char
    A0F3 992F36             362           cmpb    char, #('/')            ; '/' marks end of serial test...
    A0F6 DF15               363           be      set_diag                ; and beginning of diagnostic mode
    A0F8 995C36             364           cmpb    char, #('\')            ; '\' marks end of serial test...
    A0FB DF03E713DF         365 !         bne     exit                    ; and beginning of user mode
                            366   ;
                            367   ; This code places the board in user mode until the next RESET occurs, or
                            368   ; until RISM_STAT gets altered somehow.  It branches to a location which
                            369   ; does not get remaped, and there, a remap will be performed.
                            370   ; ---------------------------------------------------------------------------
                            371   ;
                            372           SET_BIT RISM_STAT, USER_MAP
    A103 B1FF0F             374           ldb     ioport1, #0ffh          ; reintialize ioport1
    A106 A1000118           375           ld      sp, #100H               ; clear stack
    A10A E7F3FB             376           br      user_setup
                            377   ;
                            378   $eject
```

```
ERR LOC  OBJECT          LINE       SOURCE STATEMENT
                          379
                          380    This code places the board in diagnostics mode until the next RESET or
                          381    RISM_STAT gets altered somehow.  The user's PC is loaded with the
                          382    ; address of the memory test and a 55H/0AAH pattern flashes on the
                          383    ; ioportl LEDs while the monitor is waiting for a command.
                          384    ; ----------------------------------------------------------------------
                          385    ;
    A10D                  386    set_diag:
                          387            SET_BIT RISM_STAT, DIAGNOSTIC_FLAG
    A110 A1000118         389            ld      sp, #100H              ; clear stack
    A114 A1000036         390            ld      tempw, #rism_psw       ; value for rism and initial user value
    A118 C03C36           391            st      tempw, dUSER_PSW       ; store rism psw as initial user psw
                          392    ;
    A11B A1002236         393            ld      tempw, #(mem_tst-offset)        ; Set up user pc
    A11F C03A36           394            st      tempw, dUSER_PC
                          395    ;
    A122                  396    diag_pause:
    A122 B1550F           397            ldb     ioport1, #55h
    A125                  398    diag_pause_loop:
    A125 3516FD           399            bbc     ios1,5, $              ; wait for a timer1 overflow
    A128 3516FD           400            bbc     ios1,5, $              ; ...twice
    A12B 95FF0F           401            xorb    ioport1, #0ffh         ; invert ioport1
    A12E 27F5             402            br      diag_pause_loop
                          403    ;
                          404    $eject
```

```
ERR LOC  OBJECT          LINE        SOURCE STATEMENT
                          405   ;
                          406   ; This code is executed to interpret a host command when this RISM is in
                          407   ; the diagnostics mode.
                          408   ; -----------------------------------------------------------------------
                          409   ;
    A130                  410   diag_mode:
    A130 303803E7FCDE     411 !         bbs     RISM_STAT, DLE_FLAG, force_load_data
    A136 991F36           412           cmpb    char, #1FH              ; check if byte is a command
    A139 D103E7F7DE       413 !         bh      load_data               ; commands are <= 1FH
                          414   ;
    A13E                  415   diag_command:
    A13E AC3636           416           ldbze   tempw, char             ; table lookup
    A141 643636           417           add     tempw, tempw
    A144 A3374C2136       418           ld      tempw, (diag_table-offset)[tempw]
    A149 E336             419           br      [tempw]
                          420   ;
                          421   $eject
```

```
ERR LOC  OBJECT          LINE        SOURCE STATEMENT
                          422
    A14C                  423   diag_table:
                          424   ;----------
                          425   ;
    A14C 3D00             426          dcw      (SET_DLE_FLAG     - offset)   ; 00
    A14E 1300             427          dcw      (exit             - offset)   ; 01
    A150 4200             428          dcw      (TRANSMIT         - offset)   ; 02
    A152 1300             429          dcw      (exit             - offset)   ; 03
    A154 5700             430          dcw      (READ_BYTE        - offset)   ; 04
    A156 5C00             431          dcw      (READ_WORD        - offset)   ; 05
    A158 6100             432          dcw      (READ_DOUBLE      - offset)   ; 06
    A15A 6A00             433          dcw      (WRITE_BYTE       - offset)   ; 07
    A15C 6F00             434          dcw      (WRITE_WORD       - offset)   ; 08
    A15E 7400             435          dcw      (WRITE_DOUBLE     - offset)   ; 09
    A160 7C00             436          dcw      (LOAD_ADDRESS     - offset)   ; 0A
    A162 8100             437          dcw      (INDIRECT_ADDRESS - offset)   ; 0B
    A164 B421             438          dcw      (dREAD_PSW        - offset)   ; 0C
    A166 C121             439          dcw      (dWRITE_PSW       - offset)   ; 0D
    A168 B300             440          dcw      (READ_SP          - offset)   ; 0E
    A16A CE21             441          dcw      (dWRITE_SP        - offset)   ; 0F
    A16C A621             442          dcw      (dREAD_PC         - offset)   ; 10
    A16E A121             443          dcw      (dWRITE_PC        - offset)   ; 11
    A170 7821             444          dcw      (dSTART_USER      - offset)   ; 12
    A172 8D21             445          dcw      (dSTOP_USER       - offset)   ; 13
    A174 C000             446          dcw      (REPORT_STATUS    - offset)   ; 14
    A176 4E00             447          dcw      (MONITOR_ESCAPE   - offset)   ; 15
                          448   ;
                          449   $eject
```

```
ERR LOC  OBJECT            LINE       SOURCE STATEMENT
                            450
                            451   ; The following routines, all named beginning with a 'd' for diagnostics,
                            452   ; are special cases of RISM commands used when the board is in diagnostics
                            453   ; mode.
                            454   ;
    A178                    455   dSTART_USER:
                            456   ;----------
                            457   ; Flush the pause routine off the stack and set up user's context.
                            458   ;
                            459           SET_BIT RISM_STAT, RUN_FLAG
                            461           CLR_BIT RISM_STAT, TRAP_FLAG
    A17E C701041E38         463           stb     RISM_STAT, modem_contr[0] ; update running signal to host
                            464   ;
    A183 65040018           465           add     sp,#4                  ; reset sp to overwrite RISM pc & psw,
    A187 C83A               466           push    dUSER_PC               ;   with user pc &
    A189 C83C               467           push    dUSER_PSW              ;   user psw values
                            468           EXIT_RISM
                            471   ;
    A18D                    472   dSTOP_USER:
                            473   ;---------
                            474   ; Stops "user" execution by setting up the stack to return to pause with
                            475   ; all interrupts but serial i/o locked out.
                            476   ;
    A18D CC3C               477           pop     dUSER_PSW              ; remove users psw & pc from stack
    A18F CC3A               478           pop     dUSER_PC               ;   and save
    A191                    479   dset_rism_idle:
    A191 C92221             480           push    #(diag_pause-offset)   ;   the new program counter & psw
    A194 C90000             481           push    #rism_psw
                            482           CLR_BIT RISM_STAT, RUN_FLAG
    A19A C701041E38         484           stb     RISM_STAT, modem_contr[0]     ; update running signal to host
                            485           EXIT_RISM
                            488   ;
                            489   $eject
```

```
ERR LOC  OBJECT           LINE        SOURCE STATEMENT
                           490
    A1A1                   491   dWRITE_PC:
                           492   ;--------
                           493   ; user_pc:=RISM_DATA. (Assumes user code is not running)
                           494   ;
    A1A1 C03A30            495           st      RISM_DATA, dUSER_PC
                           496           EXIT_RISM
                           499
                           500   ;
    A1A6                   501   dREAD_PC:
                           502   ;-------
                           503   ; RISM_DATA:=user_pc
                           504   ;
    A1A6 3A3805            505           bbs     RISM_STAT, RUN_FLAG, drpc_running
    A1A9 A03A30            506           ld      RISM_DATA, dUSER_PC           ; If user code is not running
                           507           EXIT_RISM
    A1AE                   510   drpc_running:
    A1AE A3180230          511           ld      RISM_DATA, 2[sp]              ; If user code is running
                           512           EXIT_RISM
                           515   ;
                           516   $eject
```

```
ERR LOC  OBJECT            LINE       SOURCE STATEMENT
                            517    ;
    A1B4                    518    dREAD_PSW:
                            519    ;---------
                            520    ; RISM_DATA:=user_psw
                            521    ;
    A1B4 3A3805             522            bbs     RISM_STAT, RUN_FLAG, drpsw_running
    A1B7 A03C30             523            ld      RISM_DATA, dUSER_PSW            ; user is not running
                            524            EXIT_RISM
    A1BC                    527    drpsw_running:
    A1BC A21830             528            ld      RISM_DATA, [sp]         ; user is running
                            529            EXIT_RISM
                            532    ;
    A1C1                    533    dWRITE_PSW:
                            534    ;---------
                            535    ; user_psw:=RISM_DATA
                            536    ;
    A1C1 3A3805             537            bbs     RISM_STAT, RUN_FLAG, dwpsw_running
    A1C4 C03C30             538            st      RISM_DATA, dUSER_PSW            ; user is not running
                            539            EXIT_RISM
    A1C9                    542    dwpsw_running:
    A1C9 C21830             543            st      RISM_DATA, [sp]         ; user is running
                            544            EXIT_RISM
                            547    ;
    A1CE                    548    dWRITE_SP:
                            549    ;--------
                            550    ; user_sp:=RISM_DATA.  (Assumes user is not running)
                            551    ;
    A1CE C01830             552            st      RISM_DATA, sp
    A1D1 27BE               553            br      dset_rism_idle
                            554    ;
                            555    $eject
```

```
ERR LOC   OBJECT           LINE        SOURCE STATEMENT
                           556
    A1D3                   557    flash_leds:
                           558    ; -------
                           559    ; On a reset this code flashes the LEDs connected to ioport1 if they are
                           560    ; enabled.  This is useful to see if the board is executing code properly.
                           561    ; If a '/' or '\' is received from the host while this routine is executing,
                           562    ; it will terminate immediately.
                           563    ; ------------------------------------------------------------------------
                           564    ;
    A1D3 A1FF0034          565            ld      rism_addr, #0FFH
    A1D7                   566    fl_wait0:
    A1D7 3516FD            567            bbc     ios1,5, fl_wait0        ; wait for a timer1 overflow
    A1DA 3516FD            568            bbc     ios1,5, $               ; ...twice
                           569    ;
    A1DD                   570    fl_loop1:
    A1DD 090134            571            shl     rism_addr, #1           ; shift another 1 into or out of
    A1E0 C40F35            572            stb     (rism_addr+1), ioport1  ; ioport1
    A1E3                   573    fl_wait1:
    A1E3 3516FD            574            bbc     ios1,5, fl_wait1        ; wait for a timer1 overflow
    A1E6 3516FD            575            bbc     ios1,5, $               ; ...twice
                           576    ;
    A1E9 88003             577            cmp     char, zero              ; check if char has been received
    A1EC D707              578            bne     quit                    ; if so exit
    A1EE 880034            579            cmp     rism_addr, zero         ; else continue flashing pattern
    A1F1 D7EA              580            bne     fl_loop1
    A1F3 27DE              581            br      flash_leds
    A1F5                   582    quit:
    A1F5 B0360F            583            ldb     ioport1, char           ; if char was received, restore it
    A1F8 F0                584            ret
                           585    ;
                           586    $eject
```

```
ERR LOC  OBJECT            LINE        SOURCE STATEMENT
    A200                   587           cseg at (offset + 2200H)
                           588   ,       ------------------------
    A200                   589   mem_tst:
                           590   ;------
                           591   ; This is a RAM test for the EV80C196KB board in its 'shipped' configuration.
                           592   ; The RAM from 2000H to 27FFH is not mapped during diagnostics, and therefore,
                           593   ; is not tested.  The test alternates between incrementing and decrementing
                           594   ; the test data on even and odd cycles of the test so that a nonrepetitive
                           595   ; pattern is produced in memory.
                           596   ; --------------------------------------------------------------------------
                           597   ;
    A200 B10116            598           ldb     iocl, #01H      ; enable PWM
    A203 011C              599           clr     ax              ; clear data register
    A205 0122              600           clr     cx              ; clear error register
    A207 011E              601           clr     dx              ; clear test count register
    A209 110F              602           clrb    ioportl
    A20B A1002820          603           ld      bx, #2800H      ; starting address of RAM in diag. mode.
    A20F                   604   loop:
    A20F C6201C            605           stb     al, [bx]        ; save test data
    A212 9A211C            606           cmpb    al, [bx]+       ; check if it is saved, and point to next byte
    A215 D71C              607           bne     failed          ; if not, test failed
                           608   ;
    A217 301E04            609           bbc     dx,0, here      ; check if test count is even or odd
    A21A 151C              610           decb    al              ; if it is odd, decrement test data
    A21C 2002              611           br      around
    A21E                   612   here:
    A21E 171C              613           incb    al              ; if it is even, increment test data
    A220                   614   around:
    A220 89008020          615           cmp     bx, #8000H      ; has end of RAM been reached by pointer?
    A224 D7E9              616           bne     loop            ; is not continue,
    A226 A1002820          617           ld      bx, #2800H      ; else, return pointer to starting address
    A22A 071E              618           inc     dx              ; count the test as successful
    A22C 170F              619           incb    ioportl         ; show completion to user on LEDs
    A22E B00F17            620           ldb     pwm_control, ioportl    ; PWM LED gets brighter as ioportl
                           621                                           ; value gets bigger
    A231 27DC              622           br      loop            ; go back for another cycle
    A233                   623   failed:
    A233 A1FFFF22          624           ld      cx, #0FFFFH     ; set error register
    A237 27FE              625           br      $               ; end test
                           626   ;
                           627   $eject
```

```
ERR LOC  OBJECT          LINE      SOURCE STATEMENT
                          628
    A280                  629            cseg at (offset + 2280H)
                          630     ;      ------------------------
    A280                  631     cycle_byte:
                          632     ;---------
                          633     ; does alternate read and write operation on the byte specified by bx.
                          634     ; -------------------------------------------------------------------
                          635            CLR_BIT IOPORT1,7
    A283                  637     cb_loop:
                          638            SET_BIT IOPORT1,7
    A286 C6201C           640            stb     ax,[bx]
                          641            CLR_BIT IOPORT1,7
    A28C B2201D           643            ldb     (ax+1),[bx]
    A28F 27F2             644            br      cb_loop
                          645     ;
    A2A0                  646            cseg at (offset + 22A0H)
                          647     ;      ------------------------
    A2A0                  648     cycle_word:
                          649     ;---------
                          650     ; does alternate read and write operation on the word specified by bx.
                          651     ; -------------------------------------------------------------------
                          652            CLR_BIT IOPORT1,7
    A2A3                  654     cw_loop:
                          655            SET_BIT IOPORT1,7
    A2A6 C2201C           657            st      ax,[bx]
                          658            CLR_BIT IOPORT1,7
    A2AC A2201E           660            ld      dx,[bx]
    A2AF 27F2             661            br      cw_loop
                          662     ;
                          663     $eject
```

```
ERR LOC  OBJECT           LINE        SOURCE STATEMENT
                           664
    9D00                   665            cseg at (offset + 1D00H)
                           666   ;        ------------------------
    9D00                   667   user_setup:
                           668   ;---------
                           669   ; This code completes changing the board into user mode.  The PLD on the
                           670   ; board (U12) automatically remaps memory when code from this address
                           671   ; range is fetched.
                           672   ; ----------------------------------------------------------------------
                           673   ;
    9D00 A1000036          674            ld      tempw, #rism_psw       ; value for rism and initial user value
    9D04 C301222036        675            st      tempw, USER_PSW        ; store rism psw as initial user psw
                           676   ;
    9D09 A1802036          677            ld      tempw, #2080H          ; Set up user pc
    9D0D C301202036        678            st      tempw, USER_PC
                           679   ;
    9D12 A13B1D36          680            ld      tempw, #(break-offset)
    9D16 C301102036        681            st      tempw, (trap-offset)[0] ; initialize trap vector
    9D1B C3013E2000        682            st      zero, (nmi-offset)[0]   ; initialize nmi vector
                           683   ;
    9D20                   684   monitor_pause:
    9D20 27FE              685            br      monitor_pause          ; wait for a command from the host
                           686   ;
                           687   $eject
```

```
ERR LOC  OBJECT              LINE      SOURCE STATEMENT
                              688
    9D22                      689   START USER:
                              690   ,---------
                              691   ; Flush the pause routine off the stack
                              692   ;
                              693           SET_BIT RISM_STAT, RUN_FLAG
                              695           CLR_BIT RISM_STAT, TRAP_FLAG
    9D28 C701041E38           697           stb     RISM_STAT, modem_contr[0]       ; update running signal to host
                              698   ;
    9D2D 65040018             699           add     sp,#4                   ; reset sp to overwrite RISM pc & psw,
    9D31 CB012020             700           push    USER_PC                 ; with user pc &
    9D35 CB012220             701           push    USER_PSW                        ; user psw values
                              702           EXIT_RISM
                              705   ;
    9D3B                      706   break:
                              707   ;----
                              708   ; This routine is invoked by a TRAP instruction used for breakpointing,
                              709   ; it operates somewhat like a STOP_USER instruction.
                              710   ;
                              711           ENTER_RISM
                              713           SET_BIT RISM_STAT, TRAP_FLAG
    9D3F 373803E74804         715 !         bbs     RISM_STAT, DIAGNOSTIC_FLAG, dSTOP_user
                              716   ;
    9D45                      717   STOP_USER:
                              718   ;----
                              719   ; Stops "user" execution by setting up the stack to return to pause with
                              720   ; all interrupts but serial i/o locked out.
                              721   ;
    9D45 CF012220             722           pop     USER_PSW                        ; remove users psw & pc from stack
    9D49 CF012020             723           pop     USER_PC                 ; and save
    9D4D                      724   set_rism_idle:
    9D4D C9201D               725           push    #(monitor_pause-offset) ; the new program counter & psw
    9D50 C90000               726           push    #rism_psw
                              727           CLR_BIT RISM_STAT, RUN_FLAG
    9D56 C701041E38           729           stb     RISM_STAT, modem_contr[0]       ; update running signal to host
                              730           EXIT_RISM
                              733   ;
                              734   $eject
```

```
ERR LOC  OBJECT            LINE      SOURCE STATEMENT
                            735   ;
    9D5E                    736   command_table:
                            737   ;------------
                            738   ;
    9D5E 3D00               739          dcw     (SET_DLE_FLAG     - offset)   ; 00
    9D60 1300               740          dcw     (exit             - offset)   ; 01
    9D62 4200               741          dcw     (TRANSMIT         - offset)   ; 02
    9D64 1300               742          dcw     (exit             - offset)   ; 03
    9D66 5700               743          dcw     (READ_BYTE        - offset)   ; 04
    9D68 5C00               744          dcw     (READ_WORD        - offset)   ; 05
    9D6A 6100               745          dcw     (READ_DOUBLE      - offset)   ; 06
    9D6C 6A00               746          dcw     (WRITE_BYTE       - offset)   ; 07
    9D6E 6F00               747          dcw     (WRITE_WORD       - offset)   ; 08
    9D70 7400               748          dcw     (WRITE_DOUBLE     - offset)   ; 09
    9D72 7C00               749          dcw     (LOAD_ADDRESS     - offset)   ; 0A
    9D74 8100               750          dcw     (INDIRECT_ADDRESS - offset)   ; 0B
    9D76 9D00               751          dcw     (READ_PSW         - offset)   ; 0C
    9D78 AC00               752          dcw     (WRITE_PSW        - offset)   ; 0D
    9D7A B300               753          dcw     (READ_SP          - offset)   ; 0E
    9D7C BA00               754          dcw     (WRITE_SP         - offset)   ; 0F
    9D7E 8D00               755          dcw     (READ_PC          - offset)   ; 10
    9D80 8600               756          dcw     (WRITE_PC         - offset)   ; 11
    9D82 221D               757          dcw     (START_USER       - offset)   ; 12
    9D84 451D               758          dcw     (STOP_USER        - offset)   ; 13
    9D86 C000               759          dcw     (REPORT_STATUS    - offset)   ; 14
    9D88 4E00               760          dcw     (MONITOR_ESCAPE   - offset)   ; 15
                            761   ;
                            762   $eject
```

```
ERR LOC  OBJECT              LINE       SOURCE STATEMENT
                              763
    8000                      764           cseg at (offset + 0000H)
                              765   ;       -----------------------
                              766   ;
                              767   ; rism interrupt service routine
                              768   ; ------------------------------
                              769   ; Control passes to this point when the rism gets a serial i/o interrupt
                              770   ; from the host system.
                              771   ;
    8000                      772   rism_isr:
                              773           ENTER_RISM
    8001 B301021E36           775           ldb     tempb, general_int[0]   ; read uart interrupt status
    8006 950436               776           xorb    tempb, #00000100B       ; test for receive fifo interrupt
    8009 DF0A                 777           be      receive_ready
    800B B10136               778           ldb     tempb, #01H             ; enable only recieve fifo interrupt...
    800E C701011E36           779           stb     tempb, gener_enabl[0]   ; of the uart, mask all others
    8013                      780   exit:
                              781           EXIT_RISM
                              784   ;
    8015                      785   receive_ready:
    8015 AF01001E36           786           ldbze   tempw, txd_rxd[0]       ; "char" is low byte of tempw
    801A 3B3803E7C420         787 !         bbc     RISM_STAT,USER_MAP, not_user
    8020 38380F               788           bbs     RISM_STAT, DLE_FLAG, force_load_data
    8023 991F36               789           cmpb    char, #1FH              ; check if byte is a command
    8026 D90D                 790           bh      load_data               ; commands are <= 1FH
                              791   ;
    8028                      792   process_command:
    8028 643636               793           add     tempw, tempw            ; convert "char" to word index
    802B A3375E1D36           794           ld      tempw, (command_table-offset)[tempw]
    8030 E336                 795           br      [tempw]
                              796   ;
                              797   $eject
```

```
ERR LOC  OBJECT          LINE        SOURCE STATEMENT
                          798
    8032                  799   force_load_data:
                          800   ;--------------
                          801           CLR_BIT RISM_STAT, DLE_FLAG
                          803   ;
    8035                  804   load_data:
                          805   ;--------
                          806   ;
    8035 0D0830           807           shll    RISM_DATA, #8          ; make room for new byte
    8038 B03630           808           ldb     RISM_DATA, char
                          809           EXIT_RISM
                          812   ;
    803D                  813   SET_DLE_FLAG:
                          814   ;-----------
                          815   ; RISM_STAT.0:=SET
                          816   ;
                          817           SET_BIT RISM_STAT, DLE_FLAG
                          819           EXIT_RISM
                          822   ;
    8042                  823   TRANSMIT:
                          824   ;-------
                          825   ; utxd:=RISM_DATA[7..0]
                          826   ; RISM_DATA:=RISM_DATA >> 8
                          827   ; RISM_ADDR:=RISM_ADDR+1
                          828   ;
                          829       SEND_DATA_BYTE
    8047 0C0830           831           shrl    RISM_DATA, #8
    804A 0734             832           inc     RISM_ADDR
                          833           EXIT_RISM
                          836   ;
    804E                  837   MONITOR_ESCAPE:
                          838   ;-------------
                          839   ; if RISM_DATA=1 then execute reset
                          840   ;
    804E 89010030         841           cmp     RISM_DATA, #01
    8052 D7BF             842           bne     exit
    8054 FF               843           rst                            ; Execute a reset instruction
    8055 27FE             844           br      $                      ; and loop until reset takes effect
                          845   ;
                          846   $eject
```

```
ERR LOC  OBJECT          LINE      SOURCE STATEMENT
                         847
    8057                 848   READ_BYTE:
                         849   ;---------
                         850   ; RISM_DATA:=byte at RISM_ADDR
                         851   ;
    8057 B23430          852          ldb     RISM_DATA, [RISM_ADDR]
                         853          EXIT_RISM
                         856   ;
    805C                 857   READ_WORD:
                         858   ;---------
                         859   ; RISM_DATA:=word at RISM_ADDR
                         860   ;
    805C A23430          861          ld      RISM_DATA, [RISM_ADDR]
                         862          EXIT_RISM
                         865   ;
    8061                 866   READ_DOUBLE:
                         867   ;-----------
                         868   ; RISM_DATA:=double_word at RISM_ADDR
                         869   ;
    8061 A23430          870          ld      RISM_DATA, [RISM_ADDR]
    8064 A3340232        871          ld      (RISM_DATA+2), 2[RISM_ADDR]
                         872          EXIT_RISM
                         875   ;
    806A                 876   WRITE_BYTE:
                         877   ;----------
                         878   ; byte at RISM_ADDR:=RISM_DATA
                         879   ; RISM_ADDR:=RISM_ADDR+1
                         880   ;
                         881      BYTE_PROTECT
    806A C63530          882          stb     RISM_DATA, [RISM_ADDR]+
                         883          EXIT_RISM
                         886   ;
    806F                 887   WRITE_WORD:
                         888   ;----------
                         889   ; word at RISM_ADDR:=RISM_DATA
                         890   ; RISM_ADDR:=RISM_ADDR+2
                         891      WORD_PROTECT
    806F C23530          892          st      RISM_DATA, [RISM_ADDR]+
                         893          EXIT_RISM
                         896   ;
    8074                 897   WRITE_DOUBLE:
                         898   ;------------
                         899   ; double-word at RISM_ADDR:=RISM_DATA
                         900   ; RISM_ADDR:=RISM_ADDR+4
                         901   ;
                         902      DWORD_PROTECT
    8074 C23530          903          st      RISM_DATA, [RISM_ADDR]+
    8077 C23532          904          st      (RISM_DATA+2), [RISM_ADDR]+
                         905          EXIT_RISM
                         908   ;
                         909   $eject
```

```
ERR LOC  OBJECT            LINE        SOURCE STATEMENT
                            910   ,
    807C                    911   LOAD_ADDRESS:
                            912   ;-----------
                            913   ; RISM_ADDR:=RISM_DATA
                            914
    807C A03034             915           ld      RISM_ADDR, RISM_DATA
                            916           EXIT_RISM
                            919
                            920   ;
    8081                    921   INDIRECT_ADDRESS:
                            922   ;---------------
                            923   ; RISM_ADDR:=[RISM_ADDR]
                            924   ;
    8081 A23434             925           ld      RISM_ADDR, [RISM_ADDR]
                            926           EXIT_RISM
                            929   ;
    8086                    930   WRITE_PC:
                            931   ;-------
                            932   ; user_pc:=RISM_DATA. (Assumes user is not running)
                            933   ;
    8086 C301202030         934           st      RISM_DATA, USER_PC
                            935           EXIT_RISM
                            938
                            939   ;
    808D                    940   READ_PC:
                            941   ;------
                            942   ; RISM_DATA:=user_pc
                            943   ;
    808D 3A3807             944           bbs     RISM_STAT, RUN_FLAG, rpc_running
    8090 A301202030         945           ld      RISM_DATA, USER_PC            ; If user code is not running
                            946           EXIT_RISM
    8097                    949   rpc_running:
    8097 A3180230           950           ld      RISM_DATA, 2[sp]              ; If user code is running
                            951           EXIT_RISM
                            954   ;
                            955   $eject
```

```
ERR LOC  OBJECT            LINE       SOURCE STATEMENT
                            956   ;
    809D                    957   READ_PSW:
                            958   ;-------
                            959   ; RISM_DATA:=user_psw
                            960   ;
    809D 3A3807             961          bbs     RISM_STAT, RUN_FLAG, rpsw_running
    80A0 A301222030         962          ld      RISM_DATA, USER_PSW            ; user is not running
                            963          EXIT_RISM
    80A7                    966   rpsw_running:
    80A7 A21830             967          ld      RISM_DATA, [sp]          ; user is running
                            968          EXIT_RISM
                            971   ;
    80AC                    972   WRITE_PSW:
                            973   ;--------
                            974   ; user_psw:=RISM_DATA (Assumes user is not running)
                            975   ;
    80AC C301222030         976          st      RISM_DATA, USER_PSW            ; user is not running
                            977          EXIT_RISM
                            980   ;
    80B3                    981   READ_SP:
                            982   ;------
                            983   ; RISM_DATA:=user_sp
                            984   ;
    80B3 4504001830         985          add     RISM_DATA, sp, #4        ; add four to account for PC and PSW...
                            986          EXIT_RISM                        ; on the stack during this interrupt
                            989   ;
    80BA                    990   WRITE_SP:
                            991   ;--------
                            992   ; user_sp:=RISM_DATA. (Assumes user is not running)
                            993   ;
    80BA C01830             994          st      RISM_DATA, sp
    80BD E78D1C             995          br      set_rism_idle
                            996   ;
    80C0                    997   REPORT_STATUS:
                            998   ;------------
                            999   ; Report user status:
      0000                 1000   stopped equ 0
      0001                 1001   running equ 1
      0002                 1002   trapped equ 2
                           1003   ;
    80C0 A1010030          1004          ld      RISM_DATA, #running
    80C4 323802274A        1005 !        bbs     RISM_STAT, RUN_FLAG, exit
    80C9 A1020030          1006          ld      RISM_DATA, #trapped
    80CD 3138022741        1007 !        bbs     RISM_STAT, TRAP_FLAG, exit
    80D2 A1000030          1008          ld      RISM_DATA, #stopped
                           1009          EXIT_RISM                        ; else report stopped
                           1012   ;
    80D8                   1013          end
```

SYMBOL TABLE LISTING

-------------------

| N A M E | VALUE | ATTRIBUTES |
|---|---|---|
| AD_COMMAND. . . . . . . . . . . . | 0002H | NULL ABS BYTE |
| AD_DONE . . . . . . . . . . . . | A002H | CODE ABS WORD |
| AD_RESULT_HI. . . . . . . . . . | 0003H | NULL ABS BYTE |
| AD_RESULT_LO. . . . . . . . . . | 0002H | NULL ABS BYTE |
| ADDR_CONTR0 . . . . . . . . . . | 1E07H | DATA ABS BYTE |
| AH. . . . . . . . . . . . . . . | 001DH | REG ABS BYTE |
| AL. . . . . . . . . . . . . . . | 001CH | REG ABS BYTE |
| AROUND. . . . . . . . . . . . . | A220H | CODE ABS ENTRY |
| AX. . . . . . . . . . . . . . . | 001CH | REG ABS WORD |
| BAUD_A_HI . . . . . . . . . . . | 1E01H | DATA ABS BYTE |
| BAUD_A_LO . . . . . . . . . . . | 1E00H | DATA ABS BYTE |
| BAUD_REG. . . . . . . . . . . . | 000EH | NULL ABS BYTE |
| BL. . . . . . . . . . . . . . . | ----- | MACRO |
| BREAK . . . . . . . . . . . . . | 9D3BH | CODE ABS ENTRY |
| BX. . . . . . . . . . . . . . . | 0020H | REG ABS WORD |
| BYTE_PROTECT. . . . . . . . . . | ----- | MACRO |
| CB_LOOP . . . . . . . . . . . . | A283H | CODE ABS ENTRY |
| CHAR. . . . . . . . . . . . . . | 0036H | REG ABS BYTE |
| CHIP_CONFIG . . . . . . . . . . | A018H | CODE ABS BYTE |
| CLOCK_CONFG . . . . . . . . . . | 1E00H | DATA ABS BYTE |
| CLR_BIT . . . . . . . . . . . . | ----- | MACRO |
| COMMAND_TABLE . . . . . . . . . | 9D5EH | CODE ABS WORD |
| CW_LOOP . . . . . . . . . . . . | A2A3H | CODE ABS ENTRY |
| CX. . . . . . . . . . . . . . . | 0022H | REG ABS WORD |
| CYCLE_BYTE. . . . . . . . . . . | A280H | CODE ABS ENTRY |
| CYCLE_WORD. . . . . . . . . . . | A2A0H | CODE ABS ENTRY |
| DEFINE_BIT. . . . . . . . . . . | ----- | MACRO |
| DIAG_COMMAND. . . . . . . . . . | A13EH | CODE ABS ENTRY |
| DIAG_MODE . . . . . . . . . . . | A130H | CODE ABS ENTRY |
| DIAG_PAUSE. . . . . . . . . . . | A122H | CODE ABS ENTRY |
| DIAG_PAUSE_LOOP . . . . . . . . | A125H | CODE ABS ENTRY |
| DIAG_TABLE. . . . . . . . . . . | A14CH | CODE ABS WORD |
| DIAGNOSTIC_FLAG . . . . . . . . | 0007H | NULL ABS |
| DLE_FLAG. . . . . . . . . . . . | 0000H | NULL ABS |
| DREAD_PC. . . . . . . . . . . . | A1A6H | CODE ABS ENTRY |
| DREAD_PSW . . . . . . . . . . . | A1B4H | CODE ABS ENTRY |
| DRPC_RUNNING. . . . . . . . . . | A1AEH | CODE ABS ENTRY |
| DRPSW_RUNNING . . . . . . . . . | A1BCH | CODE ABS ENTRY |
| DSET_RISM_IDLE. . . . . . . . . | A191H | CODE ABS ENTRY |
| DSTART_USER . . . . . . . . . . | A178H | CODE ABS ENTRY |
| DSTOP_USER. . . . . . . . . . . | A18DH | CODE ABS ENTRY |
| DUSER_PC. . . . . . . . . . . . | 003AH | REG ABS WORD |
| DUSER_PSW . . . . . . . . . . . | 003CH | REG ABS WORD |
| DWORD_PROTECT . . . . . . . . . | ----- | MACRO |
| DWPSW_RUNNING . . . . . . . . . | A1C9H | CODE ABS ENTRY |
| DWRITE_PC . . . . . . . . . . . | A1A1H | CODE ABS ENTRY |
| DWRITE_PSW. . . . . . . . . . . | A1C1H | CODE ABS ENTRY |

```
 N A M E                                      VALUE    ATTRIBUTES

DWRITE_SP . . . .                             A1CEH   CODE ABS ENTRY
DX. .                                         001EH   REG ABS WORD
ENTER_RISM.                                   -----   MACRO
EV96.                                         -----   MODULE MAIN STACKSIZE(0)
EXIT.                                         8013H   CODE ABS ENTRY
EXIT_RISM . . . . . . . . . .                 -----   MACRO
EXTERNAL_INT. . . . . . . . .                 A00EH   CODE ABS WORD
EXTERNAL_INT_PIN. . . . . . . .               A03AH   CODE ABS WORD
FAILED. . . . . . . . . . . . .               A233H   CODE ABS ENTRY
FL_LOOP1. . . . . . . . . . . .               A1DDH   CODE ABS ENTRY
FL_WAIT0. . . . . . . . . . . .               A1D7H   CODE ABS ENTRY
FL_WAIT1. . . . . . . . . . . .               A1E3H   CODE ABS ENTRY
FLASH_LEDS. . . . . . . . . . .               A1D3H   CODE ABS ENTRY
FORCE_LOAD_DATA . . . . . . . .               8032H   CODE ABS ENTRY
GENER_ENABL . . . . . . . . . .               1E01H   DATA ABS BYTE
GENERAL_INT . . . . . . . . . .               1E02H   DATA ABS BYTE
HERE. . . . . . . . . . . . . .               A21EH   CODE ABS ENTRY
HSI_DATA. . . . . . . . . . . .               A004H   CODE ABS WORD
HSI_ENTRY_4 . . . . . . . . . .               A034H   CODE ABS WORD
HSI_FIFO_FULL . . . . . . . . .               A03CH   CODE ABS WORD
HSI_MODE. . . . . . . . . . . .               0003H   NULL ABS BYTE
HSI_STATUS. . . . . . . . . . .               0006H   NULL ABS BYTE
HSI_TIME. . . . . . . . . . . .               0004H   NULL ABS WORD
HSI_ZERO. . . . . . . . . . . .               A008H   CODE ABS WORD
HSO_COMMAND . . . . . . . . . .               0006H   NULL ABS BYTE
HSO_EVENT . . . . . . . . . . .               A006H   CODE ABS WORD
HSO_TIME. . . . . . . . . . . .               0004H   NULL ABS WORD
INDIRECT_ADDRESS. . . . . . . .               8081H   CODE ABS ENTRY
INT_MASK. . . . . . . . . . . .               0008H   NULL ABS BYTE
INT_PENDING . . . . . . . . . .               0009H   NULL ABS BYTE
INVALID_OPCODE. . . . . . . . .               A012H   CODE ABS WORD
IO_MODE . . . . . . . . . . . .               1E04H   DATA ABS BYTE
IOC0. . . . . . . . . . . . . .               0015H   NULL ABS BYTE
IOC1. . . . . . . . . . . . . .               0016H   NULL ABS BYTE
IOPORT1 . . . . . . . . . . . .               000FH   NULL ABS BYTE
IOPORT2 . . . . . . . . . . . .               0010H   NULL ABS BYTE
IOS0. . . . . . . . . . . . . .               0015H   NULL ABS BYTE
IOS1. . . . . . . . . . . . . .               0016H   NULL ABS BYTE
LINE_CONFIG . . . . . . . . . .               1E03H   DATA ABS BYTE
LINE_STATUS . . . . . . . . . .               1E05H   DATA ABS BYTE
LOAD_ADDRESS. . . . . . . . . .               807CH   CODE ABS ENTRY
LOAD_DATA . . . . . . . . . . .               8035H   CODE ABS ENTRY
LOOP. . . . . . . . . . . . . .               A20FH   CODE ABS ENTRY
MEM_TST . . . . . . . . . . . .               A200H   CODE ABS ENTRY
MODEM_CONTR . . . . . . . . . .               1E04H   DATA ABS BYTE
MODEM_STATS . . . . . . . . . .               1E06H   DATA ABS BYTE
MONITOR_ESCAPE. . . . . . . . .               804EH   CODE ABS ENTRY
MONITOR_PAUSE . . . . . . . . .               9D20H   CODE ABS ENTRY
NMI . . . . . . . . . . . . . .               A03EH   CODE ABS WORD
NOT_USER. . . . . . . . . . . .               A0E4H   CODE ABS ENTRY
```

| N A M E | VALUE | ATTRIBUTES |
|---|---|---|
| OFFSET | 8000H | NULL ABS |
| PORT0 | 000EH | NULL ABS BYTE |
| PROCESS_COMMAND | 8028H | CODE ABS ENTRY |
| PWM_CONTROL | 0017H | NULL ABS BYTE |
| QUIT | A1F5H | CODE ABS ENTRY |
| READ_BYTE | 8057H | CODE ABS ENTRY |
| READ_DOUBLE | 8061H | CODE ABS ENTRY |
| READ_PC | 808DH | CODE ABS ENTRY |
| READ_PSW | 809DH | CODE ABS ENTRY |
| READ_SP | 80B3H | CODE ABS ENTRY |
| READ_WORD | 805CH | CODE ABS ENTRY |
| RECEIVE_READY | 8015H | CODE ABS ENTRY |
| REPORT_STATUS | 80C0H | CODE ABS ENTRY |
| RESET_VECTOR | A080H | CODE ABS ENTRY |
| RISM_ADDR | 0034H | REG ABS WORD |
| RISM_DATA | 0030H | REG ABS LONG |
| RISM_ISR | 8000H | CODE ABS ENTRY |
| RISM_PSW | 0000H | NULL ABS |
| RISM_STAT | 0038H | REG ABS BYTE |
| RPC_RUNNING | 8097H | CODE ABS ENTRY |
| RPSW_RUNNING | 80A7H | CODE ABS ENTRY |
| RUN_FLAG | 0002H | NULL ABS |
| RUNNING | 0001H | NULL ABS |
| SBUF | 0007H | NULL ABS BYTE |
| SEND_DATA_BYTE | ----- | MACRO |
| SERIAL_PORT | A00CH | CODE ABS WORD |
| SERIAL_RXD | A032H | CODE ABS WORD |
| SERIAL_TXD | A030H | CODE ABS WORD |
| SET_BIT | ----- | MACRO |
| SET_DIAG | A10DH | CODE ABS ENTRY |
| SET_DLE_FLAG | 803DH | CODE ABS ENTRY |
| SET_RISM_IDLE | 9D4DH | CODE ABS ENTRY |
| SOFTWARE_TIMER | A00AH | CODE ABS WORD |
| SP | 0018H | NULL ABS WORD |
| SPCON | 0011H | NULL ABS BYTE |
| SPSTAT | 0011H | NULL ABS BYTE |
| START_USER | 9D22H | CODE ABS ENTRY |
| STOP_USER | 9D45H | CODE ABS ENTRY |
| STOPPED | 0000H | NULL ABS |
| TEMPB | 0036H | REG ABS BYTE |
| TEMPW | 0036H | REG ABS WORD |
| TIMER_OVERFLOW | A000H | CODE ABS WORD |
| TIMER1 | 000AH | NULL ABS WORD |
| TIMER2 | 000CH | NULL ABS WORD |
| TIMER2_CAPTURE | A036H | CODE ABS WORD |
| TIMER2_OVERFLOW | A038H | CODE ABS WORD |
| TRANSMIT | 8042H | CODE ABS ENTRY |
| TRAP | A010H | CODE ABS WORD |
| TRAP_FLAG | 0001H | NULL ABS |
| TRAPPED | 0002H | NULL ABS |

```
 N A M E                                   VALUE    ATTRIBUTES

TXD_RXD . . . . . . . . . . . . . . .    1E00H    DATA ABS BYTE
UART. . . . . . . . . . . . . . . . .    1E00H    DATA ABS BYTE
USER_MAP. . . . . . . . . . . . . . .    0003H    NULL ABS
USER_PC . . . . . . . . . . . . . . .    2020H    DATA ABS WORD
USER_PSW. . . . . . . . . . . . . . .    2022H    DATA ABS WORD
USER_SETUP. . . . . . . . . . . . . .    9D00H    CODE ABS ENTRY
WATCHDOG. . . . . . . . . . . . . . .    000AH    NULL ABS BYTE
WORD_PROTECT. . . . . . . . . . . . .    -----    MACRO
WRITE_BYTE. . . . . . . . . . . . . .    806AH    CODE ABS ENTRY
WRITE_DOUBLE. . . . . . . . . . . . .    8074H    CODE ABS ENTRY
WRITE_PC. . . . . . . . . . . . . . .    8086H    CODE ABS ENTRY
WRITE_PSW . . . . . . . . . . . . . .    80ACH    CODE ABS ENTRY
WRITE_SP. . . . . . . . . . . . . . .    80BAH    CODE ABS ENTRY
WRITE_WORD. . . . . . . . . . . . . .    806FH    CODE ABS ENTRY
ZERO. . . . . . . . . . . . . . . . .    0000H    NULL ABS WORD

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

# Appendix D.

## Timing Analysis

## Timing analysis of the EV80C196KB board.

All values used are based on the 80C196KB operating at 12MHz. They are taken from the October 1988 version of the 80C196KB data sheet, Intel order number 270634-001.

### 80C196KB A.C. Characteristics

Tavyv = 81 ns MAX.
Tavyv(WAIT) = 11 ns (AC373 Dn to On Tplh MAX) + 35 ns (PAL/EPLD Tpd MAX)
+ 9 ns (AC08 Tplh MAX) + 12 ns (AC112 RES to Q Tphl MAX)
= 67 ns.

Tllyv is irrelevant in this design.

Tclyx = 53 ns MAX.
Tclyx(WAIT) = 10 ns (AC112 CLOCK to Q Tplh MAX).

Tllyx is irrelevant in this design.

Tavgv = 81 ns MAX.
Tclyx(BUSWIDTH) = 11 ns (AC373 Dn to On Tplh MAX) + 35 ns (PAL/EPLD Tpd MAX)
= 46 ns.

Tllgv is irrelevant in this design.

Tclgx is irrelevant in this design.

Tavdv = 183 ns MAX, for zero wait states.
Tavdv(ROMsim) = 11 ns (AC373 Dn to On Tplh MAX) + 35 ns (PAL/EPLD Tpd MAX)
+ 100 ns (RAM Tco1 MAX)
= 146 ns.

Tavdv = 349 ns MAX, for one wait state.
Tavdv(EPROM) = 11 ns (AC373 Dn to On Tplh MAX) + 35 ns (PAL/EPLD Tpd MAX)
+ 200 ns (EPROM Tce MAX)
= 246 ns.

Tavdv = 516 ns MAX, for two wait states.
Tavdv(UART) = 11 ns (AC373 Dn to On Tplh MAX) + 35 ns (PAL/EPLD Tpd MAX)
+ 288 ns (UART Tavrl MIN + Trldv MAX)
= 334 ns.

Trldv = 60 ns MAX, for zero wait states.
Trldv(ROMsim) = 50 ns (RAM Toe MAX).

Trldv = 226 ns MAX, for one wait state.
Trldv(EPROM) = 75 ns (EPROM Toe MAX).

Trldv = 393 ns MAX, for two wait states.
Trldv(UART) = 281 ns (UART Trldv MAX).

Tcldv is irrelevant in this design.

Trhdz = 63 ns MAX.
Trhdz(ROMsim) = 35 ns (RAM Tohz MAX).
Trhdz(EPROM) = 55 ns (EPROM Tdf MAX).
Trhdz(UART) = 40 ns (UART Trhdz MAX).

Trxdx = 0 ns MIN.
Trxdx(ROMsim) = 0 ns (RAM Tohz MIN).
Trxdx(EPROM) = 0 ns (EPROM Toh MIN).
Trxdx(UART) is not specified.

Txhch is irrelevant in this design.

Tclcl = 166 ns.
Tclcl(WAIT) = 55 ns (PAL/EPLD Tp MIN).
= 10 ns (AC112 1/Fmax MIN).

Tchcl = 73 ns MIN.
Tchcl(WAIT) = 25 ns (PAL/EPLD Tco MAX) + 35 ns (PAL/EPLD Tpd MAX)
+ 4 ns (AC112 Tsu MIN)
= 64 ns.
or = 25 ns (PAL/EPLD Tco MAX) + 35 ns (PAL/EPLD Tpd MAX)
+ 8 ns (AC08 Tplh MAX) + 2 ns (AC112 Trem MIN)
= 70 ns.

Tcllh is irrelevant in this design.

Tllch is irrelevant in this design.

Tlhlh is irrelevant in this design.

Tlhll = 73 ns MIN.
Tlhll(A0-A15) = 5 ns (AC373 Tw MIN).

Tavll = 68 ns MIN.
Tavll(A0-A15) = 5 ns (AC373 Ts MIN).
Tavll(WAIT) = 11 ns (AC373 Dn to On Tplh MAX) + 35 ns (PAL/EPLD Tpd MAX)
+ 8 ns (AC00 Tphl MIN) + 5 ns (AC112 Tw MIN)
= 59 ns.
Tavll(BHE#) = 11 ns (AC14 Tplh MAX) + 4 ns (AC112 Tsu MIN)
= 15 ns.

Tllax = 43 ns MIN.
Tllax(A0-A15) = 0 ns (AC373 Th MIN).
Tllax(BHE#) = 0 ns (AC112 Th MIN).

Tllrl = 43 ns MIN.
Tllrl(UART) = 7 ns (UART Tavrl MIN).

Trlcl is irrelevant in this design.

Trlrh = 411 ns MIN, for two wait states.
Trlrh(UART) = 281 ns (UART Trlrh MIN).

Trhlh = 83 ns MIN.
Trhlh(STALE) = 9 ns (74AC08 Tplh MAX) + 3 ns (74AC112 Trem MIN)
= 12 ns.

Tllwl = 73 ns MIN.
Tllwl(UART) = 7 ns (UART Tavwl MIN).

Tclwl is irrelevant in this design.

Tqvwh = 60 ns MIN, for zero wait states.
Tqvwh(ROMsim) = 40 ns (RAM Tdw MIN).

Tqvwh = 393 ns MIN, for two wait states.
Tqvwh(UART) = 90 ns (UART Tdvwh MIN).

Tchwh is irrelevant in this design.

Twlwh = 53 ns MIN, for zero wait states.
Twlwh(ROMsim) = 50 ns (RAM Twp MIN).

Twlwh = 386 ns MIN, for two wait states.
Twlwh(UART) = 231 ns (UART Twlwh MIN).

Twhqx = 73 ns MIN.
Twhqx(ROMsim) = 9 ns (74AC32 Tplh MAX) + 0 ns (RAM Tdh MIN)
= 9 ns.
Twhqx(U14) = 0 ns (RAM Tdh MIN).
Twhqx(UART) = 12 ns (UART Twhdx MIN).

Twhlh = 73 ns MIN.
Twhlh(ROMsim) = 9 ns (74AC32 Tplh MAX) + 0 ns (RAM Twr MIN)
= 9 ns.
Twhlh(UART) = 0 ns (UART Twhax MIN).
Twhlh(STALE) = 9 ns (74AC08 Tplh MAX) + 3 ns (74AC112 Trem MIN)
= 12 ns.

Twhbx is irrelevant in this design.

# Appendix E.

## Programmable Logic Equations

```
Doug Yoder
Intel
January 19, 1989
EV80C196KB 002
5AC312
Generates mapping signals for the target processor on the 80C196KB evalu-
ation board.
OPTIONS: TURBO=ON PART: 5AC312

% Input declarations %

INPUTS:  CLOCKOUT,              % MCS96 system CLOCKOUT                    %
         STALE@2,               % STretched MCS96 Address Latch Enable %
         nHLDA@3,               % 80C196KB HoLD Acknowledge                %
         A8@4,                  % MCS96 latched A8 - A15                   %
         A9@5,                  %                                          %
         A10@6,                 %                                          %
         A11@7,                 %                                          %
         A12@8,                 %                                          %
         A13@9,                 %                                          %
         A14@10,                %                                          %
         A15@11,                %                                          %
         nRESET@13              % MCS96 RESET pin                          %


% Output declarations %

OUTPUTS: nCS510@14,             % 0V => enable uart, U20                   %
         nCE2@15,               % 0V => enable U14 memory                  %
         nBUSWIDTH@16,          % 0V => put processor in 8 bit mode        %
         SB0@17,                % wait-state counter bit 0                 %
         SB1@18,                % wait-state counter bit 1                 %
         nWAIT@19,              % 0V => hold MCS96 in wait_state           %
         SB2@20,                % wait-state counter bit 2                 %
         nCE0@21,               % 0V => enable U1 and U8 memory            %
         nCE1@22,               % 0V => enable U6 and U13 memory           %
         MAP@23                 % 5V => map RAM as romsim                  %


% I/O Architecture declarations %

NETWORK:
      MAP,MAP   = RORF(MAPd,CLOCKOUT,RESET,GND,VCC)
      nWAIT     = CONF(nWAITd,VCC)
      nCS510    = COCF(UART,VCC)
      nCE2      = COCF(EEPROM,VCC)
      nCE1      = CONF(RAM,VCC)
      nCE0      = CONF(EPROM,VCC)
      nBUSWIDTH = CONF(nBWd,VCC)
```

```
% Intermediate variable definitions %

EQUATIONS:

   RESET = !nRESET;
   HLDA  = !nHLDA;

   MAPd = MAP + (RANGE3 * !STALE);

   EPROM'  = (!MAP * RANGE6)
           + RANGE1
           + RANGE4;

   RAM'    = (MAP * RANGE6)
           + RANGE7;

   EEPROM' = RANGE8;

   UART'   = RANGE5;

   OPEN0   = RANGE2
           + RANGE10;

   OPEN1   = RANGE9;

   nBWd'   = !EEPROM + !UART;

   WAIT_1  = STALE * !HLDA * (WAIT_2 + !EPROM + OPEN1);
   WAIT_2  = STALE * !HLDA * (WAIT_3 + !UART);
   WAIT_3  = WAIT_4;
   WAIT_4  = WAIT_5;
   WAIT_5  = WAIT_6;
   WAIT_6  = WAIT_7;
   WAIT_7  = GND;
   nWAITd = !WAIT;
```

```
%  Address Range Equations %

 RANGE1  = !A15 * !A14 * !A13 * !A12 * !A11 * !A10 * !A9 * !A8;  % 0000-00FF %

 RANGE2  = !A15 * !A14 * !A13 *  A12 * !A10 * !A8                 % 0100-1CFF %
         + !A15 * !A14 * !A13 * !A10 * !A9 *   A8
       + !A15 * !A14 * !A13 * !A12 *  A10
       + !A15 * !A14 * !A13 *  A11 * !A9 * !A8
       + !A15 * !A14 * !A13 *  A12 * !A11
       + !A15 * !A14 * !A13 * !A12 *  A9;

 RANGE3  = !A15 * !A14 * !A13 *  A12 * !A9                        % 1000-1DFF %
       + !A15 * !A14 * !A13 *  A12 * !A10
       + !A15 * !A14 * !A13 *  A12 * !A11;

 RANGE4  = !A15 * !A14 * !A13 *  A12 *  A11 *  A10 * !A9 *  A8;  % 1D00-1DFF %
 RANGE5  = !A15 * !A14 * !A13 *  A12 *  A11 *  A10 *  A9 * !A8;  % 1E00-1EFF %

 RANGE6  = !A15 * !A14 *  A13 * !A12 * !A11;                      % 2000-27FF %

 RANGE7  = !A15 * !A14 *  A13 *  A12                              % 2800-5FFF %
         + !A15 * !A14 *  A13 *  A11
       + !A15 *  A14 * !A13;

 RANGE8  = !A15 *  A14 *  A13;                                    % 6000-7FFF %

 RANGE9  =  A15 * !A14;                                           % 8000-BFFF %

 RANGE10 =  A15 *  A14;                                           % C000-FFFF %
```

```
% State machine %

MACHINE: WAIT_STATE
CLOCK:   CLOCKOUT
CLEAR:   RESET

STATES:        [ SB2 SB1 SB0 ] ASYNC_START  [  0   0   0  ]
     HOLD_2    [  0   0   1  ]
     HOLD_3    [  0   1   1  ]
     HOLD_4    [  1   1   1  ]
     HOLD_5    [  1   1   0  ]
     HOLD_6    [  1   0   0  ]
     HOLD_7    [  1   0   1  ]
REMOVE_HOLD    [  0   1   0  ]

ASYNC_START:    IF WAIT_1 & !WAIT_2   THEN REMOVE_HOLD
                IF WAIT_2             THEN HOLD_2
                    ASSERT: IF WAIT_1 THEN WAIT

HOLD_2:         IF WAIT_3             THEN HOLD_3
                REMOVE_HOLD
                    ASSERT:  WAIT

HOLD_3:         IF WAIT_4             THEN HOLD_4
                REMOVE_HOLD
                    ASSERT:  WAIT

HOLD_4:         IF WAIT_5             THEN HOLD_5
                REMOVE_HOLD
                    ASSERT:  WAIT

HOLD_5:         IF WAIT_6             THEN HOLD_6
                REMOVE_HOLD
                    ASSERT:  WAIT

HOLD_6:         IF WAIT_7             THEN HOLD_7
                REMOVE_HOLD
                    ASSERT:  WAIT

HOLD_7:         REMOVE_HOLD
                    ASSERT:  WAIT

REMOVE_HOLD:    ASYNC_START

END$
```

```
Name        KBBUSCON;
Partno      EV80C196KB;
Revision    01; Date        1/18/89;
Designer    Doug Yoder;
Company     Intel ECO;
Assembly    80C196KB evaluation board;
Location    U12;
Device      22V10;
/*****************************************************************/
/* Generates mapping signals for the target processor on the       */
/* 80C196KB evaluation board.                                      */
/*****************************************************************/
/* Allowable Target Device Types:   22V10                          */
/*****************************************************************/

/** Inputs **/

PIN 1 = CLOCKOUT;             /* MCS96 system CLOCKOUT                    */
PIN 2 = STALE;                /* STreched MCS96 Address Latch Enable */
PIN 3 = !HLDA;                /* 80C196KB HoLD Acknowledge                */
PIN [4..11]=[a8..a15];        /* MCS96 latched A8 - A15                   */
PIN 13 = !RESET;              /* MCS96 RESET pin                          */


/** Outputs **/

PIN 14 = !CS510;              /* 0V=> enable uart, U20                    */
PIN 15 = !CE2;                /* 0V=> enable U14 memory                   */
PIN 16 = !BUSWIDTH;           /* 0V=> put processor in 8 bit mode         */
PIN 17 = state_bit_0;         /* wait-state counter bit 0                 */
PIN 18 = state_bit_1;         /* wait-state counter bit 1                 */
PIN 19 = !WAIT;               /* 0V=> hold MCS96 in wait-state            */
PIN 20 = state_bit_2;         /* wait-state counter bit 2                 */
PIN 21 = !CE0;                /* 0V=> enable U1 and U8 memory             */
PIN 22 = !CE1;                /* 0V=> enable U6 and U13 memory            */
PIN 23 = MAP;                 /* 5V=> map ram as romsim                   */

/** Declarations and Intermediate Variable Definitions  **/

FIELD    memaddr = [a15..8];

eprom  = (!MAP & memaddr:[2000..27FF])
       #   memaddr:[0..FF] # memaddr:[1D00..1DFF];

ram    =  (MAP & memaddr:[2000..27FF]) # memaddr:[2800..5FFF];

eeprom =   memaddr:[6000..7FFF];

uart   =   memaddr:[1E00..1EFF];

open0  =   memaddr:[100..1CFF] # memaddr:[C000..FFFF];

open1  =   memaddr:[8000..BFFF];

bw     =   eeprom # uart;
```

```
wait_1  =   STALE & !HLDA & (wait_2 # eprom # open1);

wait_2  =   STALE & !HLDA & (wait_3 # uart);

wait_3  =   wait_4;

wait_4  =   wait_5;

wait_5  =   wait_6;

wait_6  =   wait_7;

wait_7  =   'b'0;

FIELD    state_count = [state_bit_0..2];

$DEFINE async_start     'b'000
$DEFINE hold_2          'b'001
$DEFINE hold_3          'b'011
$DEFINE hold_4          'b'111
$DEFINE hold_5          'b'110
$DEFINE hold_6          'b'100
$DEFINE hold_7          'b'101
$DEFINE remove_hold     'b'010

/** Wait-State Machine **/

SEQUENCE    state_count
            {
            PRESENT async_start
                    IF  wait_1  OUT     WAIT;

                    IF  wait_1  &   !wait_2 NEXT    remove_hold;
                    IF  wait_2              NEXT    hold_2;
                    DEFAULT                 NEXT    async_start;

            PRESENT hold_2
                    OUT WAIT;

                    IF  wait_3              NEXT    hold_3;
                    DEFAULT                 NEXT    remove_hold;

            PRESENT hold_3
                    OUT WAIT;

                    IF  wait_4              NEXT    hold_4;
                    DEFAULT                 NEXT    remove_hold;
```

```
        PRESENT hold_4
                OUT WAIT;

                IF  wait_5                  NEXT    hold_5;
                DEFAULT                     NEXT    remove_hold;

        PRESENT hold_5
                OUT WAIT;

                IF  wait_6                  NEXT    hold_6;
                DEFAULT                     NEXT    remove_hold;

        PRESENT hold_6
                OUT WAIT;

                IF  wait_7                  NEXT    hold_7;
                DEFAULT                     NEXT    remove_hold;

        PRESENT hold_7
                OUT WAIT;

                                            NEXT    remove_hold;

        PRESENT remove_hold
                                            NEXT    async_start;
        }
/** Logic Equations **/

MAP.D  =  (memaddr:[1000..1DFF] & !STALE) # MAP;
MAP.AR =   RESET;
MAP.SP =   'b'0;
MAP.OE =   'b'1;

state_bit_0.AR = RESET;
state_bit_0.SP = 'b'0;
state_bit_0.OE = 'b'1;
state_bit_1.AR = RESET;
state_bit_1.SP = 'b'0;
state_bit_1.OE = 'b'1;
state_bit_2.AR = RESET;
state_bit_2.SP = 'b'0;
state_bit_2.OE = 'b'1;

CE0    =   eprom;
CE1    =   ram;
CE2    =   eeprom;
CS510  =   uart;

BUSWIDTH = bw;
```

# Appendix F.

## Standard Memory-I/O Connector
## for
## EvalBoards

## General Purpose Memory Expansion Connector

Compatiblity with Other Intel Evaluation Boards

2x30 Pin Molex 39-51-2604 or Equiv.

| EV80C51FB | EV80C196KB | EV80C186 | | | EV80C186 | EV80C196KB | EV80C51FB |
|---|---|---|---|---|---|---|---|
| VCC | VCC | VCC | 1 | 2 | VCC | VCC | VCC |
| Addr 0 | Addr 0 | Addr 0 | 3 | 4 | Addr/Data 0 | Addr/Data 0 | Addr/Data 0 |
| Addr 1 | Addr 1 | Addr 1 | 5 | 6 | Addr/Data 1 | Addr/Data 1 | Addr/Data 1 |
| Addr 2 | Addr 2 | Addr 2 | 7 | 8 | Addr/Data 2 | Addr/Data 2 | Addr/Data 2 |
| Addr 3 | Addr 3 | Addr 3 | 9 | 10 | Addr/Data 3 | Addr/Data 4 | Addr/Data 4 |
| Addr 4 | Addr 4 | Addr 4 | 11 | 12 | Addr/Data 4 | Addr/Data 4 | Addr/Data 4 |
| Addr 5 | Addr 5 | Addr 5 | 13 | 14 | Addr/Data 5 | Addr/Data 5 | Addr/Data 5 |
| Addr 6 | Addr 6 | Addr 6 | 15 | 16 | Addr/Data 6 | Addr/Data 6 | Addr/Data 6 |
| Addr 7 | Addr 7 | Addr 7 | 17 | 18 | Addr/Data 7 | Addr/Data 7 | Addr/Data 7 |
| VSS | VSS | VSS | 19 | 20 | VSS | VSS | VSS |
| Addr 8 | Addr 8 | Addr 8 | 21 | 22 | Addr/Data 8 | Addr/Data 8 | N.C. |
| Addr 9 | Addr 9 | Addr 9 | 23 | 24 | Addr/Data 9 | Addr/Data 9 | N.C. |
| Addr 10 | Addr 10 | Addr 10 | 25 | 26 | Addr/Data 10 | Addr/Data 10 | N.C. |
| Addr 11 | Addr 11 | Addr 11 | 27 | 28 | Addr/Data 11 | Addr/Data 11 | N.C. |
| Addr 12 | Addr 12 | Addr 12 | 29 | 30 | Addr/Data 12 | Addr/Data 12 | N.C. |
| Addr 13 | Addr 13 | Addr 13 | 31 | 32 | Addr/Data 13 | Addr/Data 13 | N.C. |
| Addr 14 | Addr 14 | Addr 14 | 33 | 34 | Addr/Data 14 | Addr/Data 14 | N.C. |
| Addr 15 | Addr 15 | Addr 15 | 35 | 36 | Addr/Data 15 | Addr/Data 15 | N.C. |
| VSS | VSS | VSS | 37 | 38 | VSS | VSS | VSS |
| N.C. | CLKOUT | CLK | 39 | 40 | VSS | VSS | VSS |
| PSEN/RD | RD# | RD# | 41 | 42 | WR# | WR# | WR# |
| N.C./TP6 | BREQ# | ES# | 43 | 44 | BHE# | BHE# | N.C./TP4 |
| ALE | ALE | ALE | 45 | 46 | SRDY | READY | N.C./TP5 |
| N.C./TP7 | NMI | IO# | 47 | 48 | DRQ0 | INST | RD# |
| RESET# | RESET# | RESET | 49 | 50 | INT0 | EXTINT/P2.2 | INT0/P3.2 |
| PAL Disable# | Note 2 | T0OUT | 51 | 52 | T0IN | N.C. | PSEN# |
| N C | HLDA# | HLDA | 53 | 54 | HOLD | HOLD# | N.C. |
| 12VDC | -12VDC | -12V | 55 | 56 | +12VDC | +12VDC | +12VDC |
| VSS | VSS | VSS | 57 | 58 | VSS | VSS | VSS |
| VCC | VCC | VCC | 59 | 60 | VCC | VCC | VCC |

Note 1

N.C = No Connect

N.C./TPx = *No Connect, but routed to an on-board test point for the user.*

Note 2:

Pin 51 of the EV80C196KB will be connected to U12 pin 20 on future revisions of this board.

# Appendix G.

## Sample Session

This list file was produced by using the command "list demo.lst" before invoking demo.log with the command "include demo.log" as described below. This list file can be used to compare to the screen of your own PC while you are running demo.log.

```
;===List file opened on 01/24/1989 at 16:43:15
*include demo.log
;---INCLUDE FILE OPEN
*;
*; This is a demo of some of the features of iECM-96 for use with the
*; EV80C196KB board.  In order to run the demo, place the software disk in
a
*; drive.  Then select that drive by typing "A:" or "B:", whichever core-
sponds
*; to that drive, and a carriage return.  Type "ECM96" and carriage re-
turn.
*; At the asterisk prompt type "INCLUDE DEMO.LOG" and carriage return.
*;
*; For additional information, please see the EV80C196KB Microcontroller
*; Evaluation Board USER'S MANUAL.
*;
*pause
;   Hit the space bar to continue...
*;
*; This command loads 96KBDEMO.OBJ from disk.
*;
*load 96kbdemo.obj
;
;   mod name is: |DFMO96KB|
;   mod date stamp is: 01/24/89 16:34:47
*;
*pause
;   Hit the space bar to continue...
*;
*dasm 2080,8 ; This disassembles 8 lines of code starting at 2080H
;                           | RESET_VECTOR:
; 2080: A1000118            |     LD        18,#0100
; 2084: 011C                |     CLR       AX
; 2086: 0120                |     CLR       CX
; 2088: 0122                |     CLR       DX
; 208A: B10116              |     LDB       16,#01
; 208D: 110F                |     CLRB      IOPORT1
; 208F: 1117                |     CLRB      17
; 2091: A1BF201E            |     LD        BX,#20BF
*pause
;   Hit the space bar to continue...
*;
*pc ; This displays the current value of the Program counter.
;   PC=RESET_VECTOR
*;
```

```
*; To change the Program Counter use "pc = 2080<cr>".
*;
*pause
;   Hit the space bar to continue...
*;
*go from 2080 forever ; This command clears all breakpoints and executes
code.
>;
>; The LED's for I/O Port 1 should be incrementing regularly.
>;
>pause
;   Hit the space bar to continue...
>;
>dasm .past,8 ; The disassmbler and all other memory read commands can
be....
;                        | PAST:
; 20A6: 8900801E         |    CMP     BX,#8000
; 20AA: D7E9             |    JNE     LOOP
; 20AC: A1BF201E         |    LD      BX,#20BF
; 20B0: 0722             |    INC     DX
; 20B2: 170F             |    INCB    IOPORT1
; 20B4: B00F17           |    LDB     17,IOPORT1
; 20B7: 27DC             |    SJMP    LOOP
;                        | FAILED:
; 20B9: A1FFFF20         |    LD      CX,#0FFFF
>;
>;               used while code is running on the board.
>;
>pause
;   Hit the space bar to continue...
>;
>asm 20b2 ; start assembling code at address 20b2H, see disassembly list-
ing.
;   Single Line Assembler activated, exit with "end" directive
; 20B2H: decb .ioport1
; 20B4H: end
>pause
;   Hit the space bar to continue...
>;
>; The LED's for I/O Port 1  should now be decrementing.
>;
>; Note that not only is there an assembler, it and all other memory modi-
fing
>; commands can be used while the board is executing user code.  However,
use
>; caution when modifing code while it is running, the resulting code may
>; cause errors due to variable length instructions.
>;
>pause
```

```
;   Hit the space bar to continue...
>;
>halt
*dasm .loop,9
;                          | LOOP:
; 2095: C61E1C             |    STB     AL,[1E]
; 2098: 9A1F1C             |    CMPB    AL,[1E]+
; 209B: D71C               |    JNE     FAILED
;                          | HERE:
; 209D: 382204             |    JBS     22,00,BACK
; 20A0: 171C               |    INCB    AL
; 20A2: 2002               |    SJMP    PAST
;                          | BACK:
; 20A4: 151C               |    DECB    AL
;                          | PAST:
; 20A6: 8900801E           |    CMP     BX,#8000
; 20AA: D7E9               |    JNE     LOOP
*pause
;   Hit the space bar to continue...
*;
*go from 2080 till 20a6 ; This go command sets a breakpoint[0] = 20a6H.
*pause
;   Hit the space bar to continue...
*;
*pc ; Code has stopped at the breakpoint!  Note that 20a6 has not executed
yet.
;   PC=PAST
*pause
;   Hit the space bar to continue...
*;
*br ; This command displays all breakpoints, 20a6 has been set.
;   BREAKPOINT[0] = PAST
*pause
;   Hit the space bar to continue...
*;
*br[0]=0 ; This command clears breakpoint[0].
*pause
;   Hit the space bar to continue...
*;
*br ; As can be shown.
;   NO BREAKPOINTS ARE ACTIVE
*pause
;   Hit the space bar to continue...
*;
*br[0f]=20a6 ; This command sets breakpoint[15] = 20a6.
*pause
;   Hit the space bar to continue...
*;
*br ; See?
;   BREAKPOINT[15]= PAST
*pause
```

```
;   Hit the space bar to continue...
*;
*; This concludes the demo, we hope you enjoy using the EV80C196KB board.
*;
*pause
;   Hit the space bar to continue...
*;
*; Type "QUIT" and carriage return to exit iECM-96.
*;
*quit
```

# NORTH AMERICAN SALES OFFICES

**ALABAMA**

Intel Corp.
600 Boulevard South
suite 104-L
Huntsville 35802
Tel: (205) 883-3507
FAX: (205) 883-3511

**ARIZONA**

†Intel Corp.
410 North 44th Street
Suite 500
Phoenix 85008
Tel: (602) MI-0388
FAX: (602) 2440448

CALIFORNIA

†Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500
FAX: (818) 340-I 144

Intel Corp.
1 Sierra Gate Plaza
Suite 280C
Roseville 95578
Tel: (918) 782-8086
FAX: (918) 782-8153

†Intel Corp.
9665 Chesapeake Dr.
Suite 325
San Diego 92123
Tel: (619) 292-8086
FAX: (619) 292-0628

*†Intel Corp.
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642
TWX: 91 o-595-1 114
FAX: (714) 541.9157

*†Intel Corp.
San Tomas 4
2700 San Tomas Expressway
2nd Floor
Santa Clara 95051
Tel: (408) 986-8086
TWX: 91 o-338-0265
FAX (408) 727-2620

**COLORADO**

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (719) 594-6622
FAX: (303) 594-0720

*†Intel Corp
600 S. Cherry St
Suite 700
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289
FAX: (303) 322-8670

**CONNECTICUT**

†Intel Corp
301 Lee Farm Corporate Park
83 Wooster Heights Rd
Danbury 08610
Tel: (203) 748-3130
FAX: (203) 794-0339

FLORIDA

†Intel Corp.
800 Fairway Drive
Suite 160
Deerfield Beach 33441
Tel: (305) 421-0506
FAX: (305) 421.2444

†Intel Corp.
5850 T.G. Lee Blvd.
Suite 340
Orlando 32822
Tel: (407) 240-8000
FAX: (407) 240-8097

GEORGIA

†Intel Corp.
20 Technology Parkway
Suite 150
Norcross 30092
Tel: (404) 449-0541
FAX: (404) 605-9762

ILLINOIS

*†Intel Corp.
Woodfield Corp. Center III
300 N. Martingale Road
Suite 400
Schaumburg 60173
Tel: (708) 605-8031
FAX: (706) 706-9762

INDIANA

†Intel Corp.
8910 Purdue Road
Suite 350
Indianapolis 46268
Tel: (317) 875-0623
FAX: (317) 875-8938

**MARYLAND**

*†Intel Corp.
10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (410) 206-2860
FAX: (410) 206-3678

**MASSACHUSETTS**

● †Intel Corp.
Westford Corp. Center
3 Carlisle Road
2nd Floor
Westford 01888
Tel: (508) 692-0960
TWX 710-343-6333
FAX: (508) 692-7867

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48322
Tel: (313) 851-8096
FAX: (313) 851-8770

**MINNESOTA**

†Intel Corp.
3500 W. 80th St.
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867
FAX: (812) 631-6497

NEW JERSEY

*†Intel Corp.
Lincroft Office Center
125 Hall Mile Road
Red Bank 07701
Tel: (908) 747-2233
FAX: (908) 747-0983

NEW YORK

*Intel Corp.
850 Crosskeys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391
FAX: (716) 223-2561

*†Intel Corp.
2950 Express Dr.. South
Suite 130
Islandia 11722
Tel: (516) 231-3300
TWX: 510-227-6236
FAX: (516) 348-7939

†Intel Corp.
300 Westage Business Center
Suite 230
Fiihkill 12624
Tel: (914) 897-3860
FAX: (914) 897.3125

OHIO

*†Intel Corp.
3401 Park Center Drive
suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 61 O-450-2526
FAX: (513) 890-8658

*†Intel Corp.
25700 Science Park Dr.
Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 81 O-427-9296
FAX: (604) 282-0673

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma Cii 73182
Tel: (405) 848-8086
FAX: (405) 840-9819

**OREGON**

†Intel Corp.
15254 N.W. Greenbrier Pkwy.
Building B
Beaverton 97006
Tel: (503) 8458051
TWX 910-467-8741
FAX: (503) 6458181

**PENNSYLVANIA**

*†Intel Corp.
925 Harvest Drive
Suite 200
Blue Bell 19422
Tel: (215) 641-1000
FAX: (215) 641-0785

*†Intel Corp.
400 Penn Center Blvd
Suite 610
Pittsburgh 15235
Tel: (412) 8234970
FAX: (412) 829.7578

PUERTO RICO

†Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

SOUTH CAROUNA

Intel Corp.
100 Executive Center Drive
Suite 109
Greenville 29815
Tel: (803) 297-8086
FAX: (803) 297-3401

**TEXAS**

†Intel Corp.
8911 N. Capital of Texas Hwy.
Suite 4230
Austin 78759
Tel: (512) 794-8086
FAX: (512) 338-9335

*†Intel Corp.
12000 Ford Road
suite 400
Dallas 75234
Tel: (214) 241-8087
FAX: (214) 484-1180

● †Intel Corp.
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX 91 O-881 -2490
FAX: (713) 9663660

**UTAH**

†Intel Corp.
426 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 266.1457

WASHINGTON

†Intel Corp.
2800 156th Avenue S.E.
Suite 105
Bellevue 98008
Tel: (206) 643-8086
FAX: (206) 746-4495

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086
FAX: (509) 926-9467

**WISCONSIN**

Intel Corp.
400 N. Executive Dr.
Suite 401
Brookfield 53005
Tel: (414) 789-2733

## CANADA

BRITISH COLUMBIA

Intel Semiconductor of
Canada, Ltd.
4585 Canada Way
Suite 202
Bumaby V5G 4L6
Tel: (604) 298-0387
FAX: (604) 298-8234

ONTARIO

†Intel Semiconductor of
Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: (613) 820.5936

†Intel Semiconductor of
Canada. Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
FAX: (418) 875-2438

**QUEBEC**

†Intel Semiconductor of
Canada, Ltd.
1 Rue Holiday
Suite 115
Tour East
Pt. Claire H9R 5N3
Tel: (514) 694-9130
FAX: 514-694-0064

†Sales and Service Office
*Field A&cation Location

CG/SALE/021492

# NORTH AMERICAN SERVICE OFFICES

ALASKA

Intel Corp.
c/o TransAlaska Network
1515 Lore Rd.
Anchorage 99507
Tel: (907) 522-1776

Intel Corp.
c/o TransAlaska Date Systems
do GCI Operations
520 Fifth Ave., Suite 407
Fairbanks 99701
Tel: (907) 4526264

**ARIZONA**

*Intel Corp.
410 North 44th Street
Suite 500
Phoenix 85008
Tel: (602) 2316386
FAX: (602) 2446446

*Intel Corp.
500 E. Fry Blvd., Suite M-15
Sierra Vista 85635
Tel: (602) 459-5010

**ARKANSAS**

Intel Corp.
c/o Federal Express
1580 West Park Drive
Little Rock 72204

CALIFORNIA

*Intel Corp.
21515 Vanowen St.. Ste.116
Canoga Park 91303
Tel: (816) 704-8500

*Intel Corp..
300 N. Continental Blvd.
Suite 100
El Segundo 90245
Tel: (213) 640-6040

'Intel Corp.
1900 Prairie City Rd.
Folsom 95630-9597
Tel: (916) 3516143

'Intel Corp.
9665 Chesapeake Dr.; Suite 325
San Diego 92123
Tel (619) 2928086

**Intel Corp.
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel (714) 8359642

**Intel Corp
2700 San Tomas Exp., 1st Floor
Santa Clara 95051
Tel: (408) 970-1747

COLORADO

'Intel Corp
600 S. Cherry s l Suite 700
Denver 80222
Tel: (303) 321-8086

**CONNECTICUT**

*Intel Corp
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 08611
Tel: (203) 748-3130

FLORIDA

**Intel Corp.
800 Fairway Dr., Suite 160
Deerfield Beach 33441
Tel: (305) 4218506
FAX: (305) 421-2444

*Intel Corp.
5850 T.G. Lee Blvd., Ste. 340
Orlando 32822
Tel: (407) 240-8000

**GEORGIA**

*Intel Corp.
20 Technology Park, Suite 150
Norcross 30092
Tel: (404) 4490641

5523 Theresa Street
Columbus 31907

HAWAII

**Intel Corp.
Honolulu 98620
Tel: (808) 8476738

ILLINOIS

• *Tintel Corp.
Woodfield Corp. Center III
300 N. Martingale Rd., Ste. 400
Schaumburg 60173
Tel: (708) 8058031

INDIANA

• 1nte( Corp.
8910 Purdue Rd., Ste. 350
Indiinapolii 46288
Tel: (317) 8758823

KANSAS

*Intel Corp.
10985 Cody, Suite 140
Overland Park 66210
Tel: (913) 345.2727

KENTUCKY

Intel Corp.
133 Walton Ave.. Office 1A
Lexington 40508
Tel: (606) 2552957

Intel Corp.
896 Hillcrest Road Apt. A
Radcliff 40160 (Louisville)

**LOUISIANA**

Hammond 70401
(serviced from Jackson, MS)

MARYLAND

**Intel Corp.
10010 Junction Dr., Suite 200
Annapolis Junction 20701
Tel: (361) 206-2860

**MASSACHUSETTS**

**Intel Corp.
Westford Corp. Center
3 Carlisle Rd., 2nd floor
Westford 01868
Tel: (608) 692-0960

MICHIGAN

*Intel Corp.
7071 Orchard Lake Rd.. Ste. 100
West Bloomfield 46322
Tel: (313) 851-8905

**MINNESOTA**

*Intel Corp.
3500 W. 80th St., Suite 360
Bloomington 55431
Tel: (612) 8358722

MISSISSIPPI

Intel Corp.
c/o Compu-Care
2001 Airport Road, Suite 205F
Jackson 39208
Tel: (601) 932-6275

MISSOURI

*Intel Corp.
3300 Rider Trail South
Suite 170
Earth City 63045
Tel: (314) 291-1990

Intel Corp.
Route 2, Box 221
Smithville 84089
Tel: (913) 3452727

NEW JERSEY

**Intel Corp.
300 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0821

*Intel Corp.
Lincroft Office C-enter
125 Half Mile Road
Red Bank 07701
Tel: (908) 747-2233

NEW **MEXICO**

Intel Corp.
Rio Ranch0 1
4100 Sara Road
A l Rancho 87124-1025
(near Albuquerque)
Tel: (505) 893-7000

NEW YORK

*Intel Corp.
2950 Expressway Dr. South
Suite 130
Islandia 11722
Tel: (516) 2313300

Intel Corp.
300 Westage Business Center
Suite 230
Fishkill 12524
Tel: (914) 8973860

Intel Corp.
5858 East Molloy Road
Syracuse 13211
Tel: (315) 4546576

NORTH CAROLINA

*Intel Corp.
5800 Executive Center Drive
Suite 105
Charlotte 28212
Tel: (704) 5888966

**Intel Corp.
5540 Centerview Dr., Suite 215
Raleigh 27666
Tel: (919) 851-9537

**OHIO**

"Intel Corp.
3401 Park Center Dr., Ste. 220
Dayton 45414
Tel: (513) 890-5350

*Intel Corp.
25700 Science Park Dr., Ste. 100
Beachwood 44122
Tel: (218) 464-2736

**OREGON**

**Intel Corp.
15254 N.W. Greenbrier Pkwy.
Building B
Beaverton 97008
Tel: (503) 645-8051

PENNSYLVANIA

• Tintel Corp.
925 Harvest Drive
Suite 200
Blue Bell 19422
Tel: (215) 641-1000
1-800-468-3548
FAX: (215) 641-0785

• YIntel Corp.
400 Penn Center Blvd.. Ste. 610
Pittsburgh 15235
Tel: (412) 8234970

'Intel Corp.
1513 Cedar Cliff Dr.
Camp Hill 17011
Tel: (717) 761-0860

PUERTO RICO

Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (609) 7338816

TEXAS

"Intel Corp.
Westech 360, Suite 4230
8911 N. Capitol of Texas Hwy.
Austin 78752-1239
Tel : (512) 794-8086

**†Intel Corp.
12000 Ford Rd., Suite 401
Dallas 75234
Tel: (214) 2418087

**Intel Corp.
7322 SW Freeway, Suite 1490
Houston 77074
Tel: (713) 9888066

UTAH

Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 268-1457

**VIRGINIA**

*Intel Corp.
9030 Stony Point Pkwy
Suite 360
Richmond 23235
Tel: (804) 330-9393

**WASHINGTON**

**Intel Corp.
155 106th Avenue N.E.. Ste. 386
Bellevue 98094
Tel: (206) 4538086

## CANADA

ONTARIO

**Intel Semiconductor of
Canada, Ltd.
2650 Queensview Dr.. Ste. 250
Ottawa K2B 8H6
Tel: (613) 6299714

**Intel Semiconductor of
Canada, Ltd.
190 Attwell Dr., Ste. 102
Rexdale (Toronto) M9W 6H8
Tel: (416) 675-2105

**QUEBEC**

**Intel Semiconductor of
Canada, Ltd.
1 Rue Holiiay
Suite 115
Tour East
Pt. Claire H9R 5N3
Tel: (514) 694-9130
FAX: 514-694-0064

# CUSTOMER TRAINING CENTERS

**ARIZONA**

2402 W. Beardsley Road
Phoenix 85027
Tel: (602) 869-4288
1X0-466-3548

# SYSTEMS ENGINEERING OFFICES

MINNESOTA

3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (812) 835-6722

**NEW YORK**

2950 Expressway Dr., South
Islandia 11722
Tel: (506) 2313300

*Carry-in locations
"Carry-in/marl-in locations

**UNITED STATES**
Intel Corporation
2200 Mission College Blvd.
P.O. Box 58119
Santa Clara, CA 95052-8119

**JAPAN**
Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki-ken 300-26

**FRANCE**
Intel Corporation S.A.R.L.
1, Quai de Grenelle
75015 Paris

**UNITED KINGDOM**
Intel Corporation (U.K.) Ltd.
Pipers Way, Swindon
Wiltshire, England SN3 1RJ

**GERMANY**
Intel GmbH
Dornacher Strasse 1
85622 Feldkirchen/Muenchen

**HONG KONG**
Intel Semiconductor Ltd.
32/F Two Pacific Place
88 Queensway, Central

**CANADA**
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

**Intel embedded architectures and flash memory are supported by an array of development tools solutions. Use the World Wide Web, FaxBack, Literature Centers, and Intel Hotline for comprehensive tools information.**

Printed in USA/297 100/IL HS