

**APPLICATION  
NOTE**

**Add-in Flash Memory  
Card for an 80C186EA  
Application**

**DR. MAHESH RAO**  
APPLICATIONS SUPPORT GROUP, ASMO  
**DAVID BOUD**  
APPLICATIONS SUPPORT GROUP, ASMO  
**ANDREW GAFKEN**  
FLASHCARD DESIGN GROUP, MCD

September 1994

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

MDS is an ordering code only and is not used as a product name or trademark of Intel Corporation.

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

\*Other brands and names are the property of their respective owners.

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation  
Literature Sales  
P.O. Box 7641  
Mt. Prospect, IL 60056-7641  
or call 1-800-879-4683



# CONTENTS

	PAGE		PAGE
1.0 ABSTRACT.....	1	5.0 SOFTWARE.....	4
2.0 INTRODUCTION.....	1	6.0 CONCLUSION.....	10
3.0 PCMCIA-186EA INTERFACE.....	1	<b>APPENDIX A</b>	
4.0 HARDWARE.....	2	<b>MONITOR AND CONTROL PROGRAM</b>	
4.1 Timing Analysis.....	3	<b>FOR 186EA-PCMCIA INTERFACE</b>	
4.2 PCIC Use.....	4		
4.3 PIA Use.....	4		



## 1.0 ABSTRACT

This paper discusses an application of Series 2 Flash memory cards in an embedded 80C186 design. The Series 2 Flash memory cards could be used for data as well as code storage. This paper describes a generalized, low-cost design approach to interfacing Series 2 Flash memory cards with embedded 80C186-based systems to vastly enhance the system's data storage capability, as well as allowing designers to broaden the scope of their applications.

## 2.0 INTRODUCTION

Engineers face a growing need to produce ROMable code for Intel architecture microprocessors. The low cost, high quality, and availability of standard DOS tools make them attractive candidates for ROMable-code generators [1]. Even though most 80C186 embedded systems do not run under DOS, using DOS-based PCs to develop, test, debug, and port ROMable code would make development easier and cheaper and could considerably reduce time to market. Using Intel Flash memory cards in an 80C186-based embedded system would address the issue of porting the code as well as increasing the addressable memory space [2]. A simple implementation using Intel Flash memory cards in an 80C186-based system is discussed here.

## 3.0 PCMCIA-186EA INTERFACE

The 80C186 family of embedded processors are highly integrated, 16-bit processors of choice for high-performance embedded applications. The 80C186EA is the second member of the 80C186 modular core family. It is 100% code-compatible with the standard 80C186, but includes two power management modes. Intel offers an 8-bit bus version, called the 80C188EA, as well as 3-volt versions designated the 80L186EA and 80L188EA. Upgrades from the standard 80C186 applications to the 80C186EA microprocessor require no software modifications and, in most cases, little or no hardware modification. The details of upgrade are provided in AP-468 [3].

Intel's Series 2 Flash memory cards facilitate high-performance disk emulation in mobile PCs and dedicated equipment [4]. Series 2 Flash memory cards conform to

the Personal Computer Memory Card International Association (PCMCIA 2.0)/Japanese Electronics Industry Development Association (JEIDA 4.1) 68-pin standard, providing electrical and physical compatibility. In addition, the Series 2 Flash memory card is compatible with Intel's Exchangeable Card Architecture (ExCA™ architecture), an open hardware and software system implementation of PCMCIA Release 2.0 that allows portability from system to system, independent of manufacturer.

The Flash memory card's low power consumption, transportability, and compatibility make it an ideal choice for designers in the embedded market. Potential embedded applications include portable or handheld controllers, low-power data acquisition and control systems (DAQM's), systems in harsh environments, and systems that would benefit from solid-state removable memory, to mention just a few. The targeted embedded markets, in general, are those in which addressable memory has been a constraint or those that need to port data to other applications. The 80C186 embedded processor family is being used as the core processor in many embedded applications; however, the 80C186 can directly address a maximum space of 1 Mbyte, which may limit its usage in some applications. The addition of Flash memory cards would broaden the application base of these processors. Along with the power-saving features of the 80C186EA and availability of the 3.3-volt powered 80L186EA, the power saving features of the Series 2 Flash memory cards would definitely find wider application bases in embedded and industrial engineering.

In this example, a simple paging scheme was used to increase the memory addressability of the 80C186-based platform. Page length was fixed to 64 Kbytes. The reasons for using the paging method are simplicity and ease. Also, 64 Kbytes is equal to the segmentation value in 80C186-based systems as well as the page length of the Series 2 Flash memory card. This makes it very attractive to have pages of 64 Kbytes each. The additional overhead for setting up pages is very small compared to the additional space for data storage or code execution, and page set-up changes would be needed only when going to the next 64 Kbyte boundary. For smooth code execution, paging may not be the best solution; however, for data storage, paging is one of the best solutions available.

### 4.0 HARDWARE

The described add-in board increases the addressable memory space of both existing and future 80C186 platforms, with minor modifications in some cases. Since the add-in board would be based on Series 2 Flash memory cards, the interface would meet the PCMCIA standard. Meeting the PCMCIA standard would increase the 80C186 memory space availability from 1 Mbyte to as high as 64 Mbytes. This add-in board would allow designers to develop, test, and debug software on a DOS-based PC, then download the software to the embedded system using Flash memory cards. In addition, Flash memory cards offer these features:

- They are nonvolatile, so they do not require a battery backup for data retention as SRAM memory subsystems typically do.
- Their power-saving modes allow for power conservation strategies in various operating conditions.
- They can be removed from the field unit and transferred to computer systems for data retrieval and analysis.
- They make it possible to change the operating software.

Figure 1 shows a block diagram of the system and Figure 2 shows the map of memory and input/output devices.

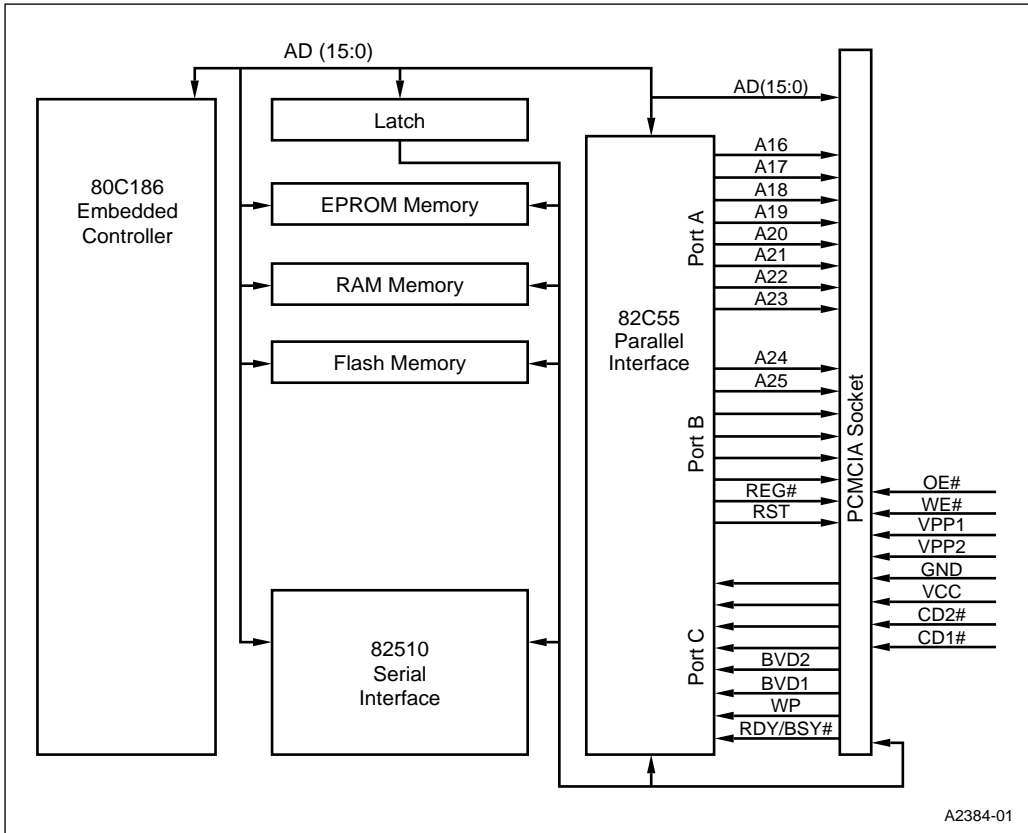
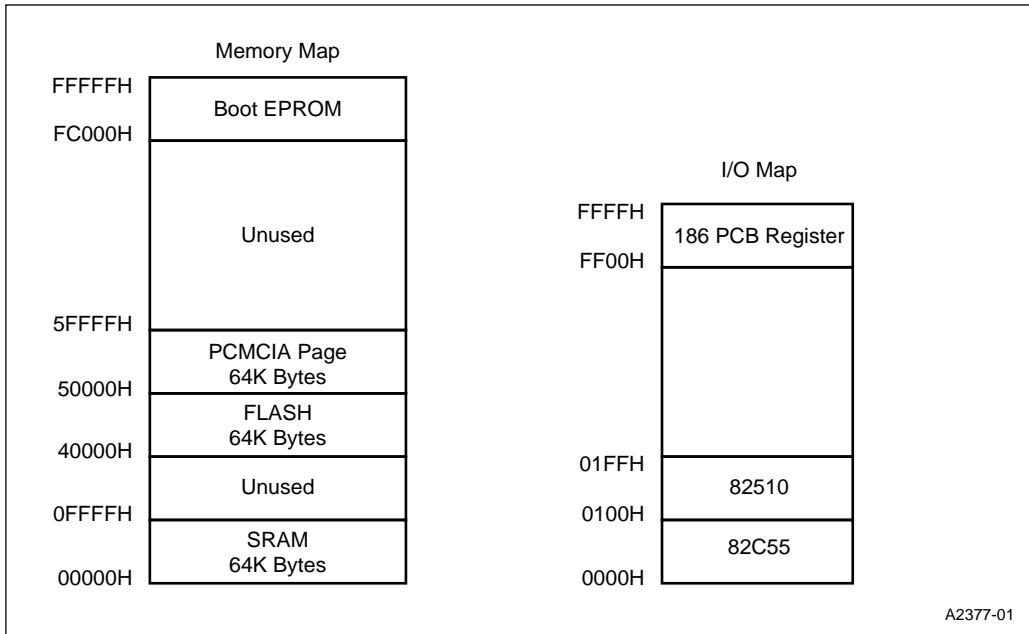


Figure 1. System Block Diagram



A2377-01

**Figure 2. Memory and I/O Device Map**

The system uses an 82510 serial interface for communicating with the outside world and an 82C55 peripheral interface adapter for memory paging. The system has the following major components:

- 64 Kbytes of SRAM
- 16 Kbytes of program memory (EPROM)
- 64 Kbytes of Flash memory
- a PCMCIA 2.0 interface with addressing capabilities for cards up to 64 Mbytes
- an 82510 serial interface for RS-232C-based communication capabilities
- an 82C55 programmable peripheral interface adapter (PIA)

## 4.1 Timing Analysis

Series 2 Flash memory cards are offered with various read access times ranging from 150 ns to 250 ns. These times are referenced only to the read cycle; write cycle times are

typically longer than read cycle times. The 80C186EA is offered in frequencies of up to 20 MHz, resulting in clock periods of 50 ns ( $T=50$  ns).

Output enable (RD#) to data valid on the Series 2 Flash memory cards is represented by  $T_{GLOV}$ , which is a maximum of 100 ns and should be less than  $(2+n)T - T_{CLOV2} - T_{CLIS}$  of the 80C186EA; larger values would require wait states. Also, the address valid to data valid time for the 80C186EA is  $(3+n)T - T_{CLOV2} - T_{ADLTCH} - T_{CLIS}$ , which should be greater than the card's 250 ns address access time,  $T_{AVOV}$ . In the equations above,  $n$  is the number of wait states to be inserted. With the insertion of each wait state, the 80C186EA's timing would increase by a time equal to one  $T$  (50 ns at 20 MHz).

An 80C186EA operating at 20 MHz would require a minimum of three wait states to guarantee reliable read operations for cards with up to 250 ns read access time. It would require one wait state for read operations even for cards with 150 ns read access time. The same number of wait states would also work for write operations because the 80C186EA polls the card to see whether it is ready for the next write.

## 4.2 PCIC Use

Intel's PC Card Interface Controller (PCIC) is the ExCA™ architecture interface solution for notebook PCs [5]. The 82365SL allows PC manufacturers to design systems that provide a wide range of connection or communication options (modem, Twisted Pair Ethernet, etc.) as well as eliminating rotating electromechanical media (via Intel Flash memory cards). However, the 82365SL was not used in this design, for the following reasons:

- The scope of the project was to interface Flash memory cards to an 80C186-based system, and cost was one consideration. The cost of the 82365SL is greater than that of the 82C55.
- This design would not use the 82365SL's full functionality and capabilities, so the increased system cost would not be justified.
- The complexity of the design using the 82365SL would be greater, in both hardware and software, than that of the proposed design.

## 4.3 PIA Use

This design implements a solution that is both cost-effective and easy to implement. The design uses multiple ports to set up pages of 64 Kbytes each. The primary page can be mapped to any available 64 Kbyte segment of the 80C186 memory space and the consecutive pages can be set up using the PIA. The parallel interface chosen for this application is Intel's 82C55 Programmable Peripheral Interface Adapter [6]. This CMOS device is inexpensive and easy to program, making it a cost-effective solution. However, using the 82C55 imposes some limitations on the proposed system:

- It limits the maximum page size to 64 Kbytes.
- It requires special memory management software to manage the page frames.

## 5.0 SOFTWARE

Once the interface hardware was built, firmware was written to test and control the system functionality. This software allows the user to run various tests using a PC or laptop computer. Software initializes the system components such as the 82C55 and 82510 as well as the

80C186EA. Figure 4 shows a flowchart of the system initialization, and Figure 3 shows the interactive menu portion of the software.

```

WELCOME TO PCMCIA-186 INTERFACE
TO CHECK SYSTEM FUNCTIONALITY,
ENTER ANY OF THE FOLLOWING CHOICES.
1. RAM TEST
2. READING FROM PCMCIA CARD
3. WRITING TO PCMCIA CARD
4. EXIT

```

**Figure 3. System Menu**

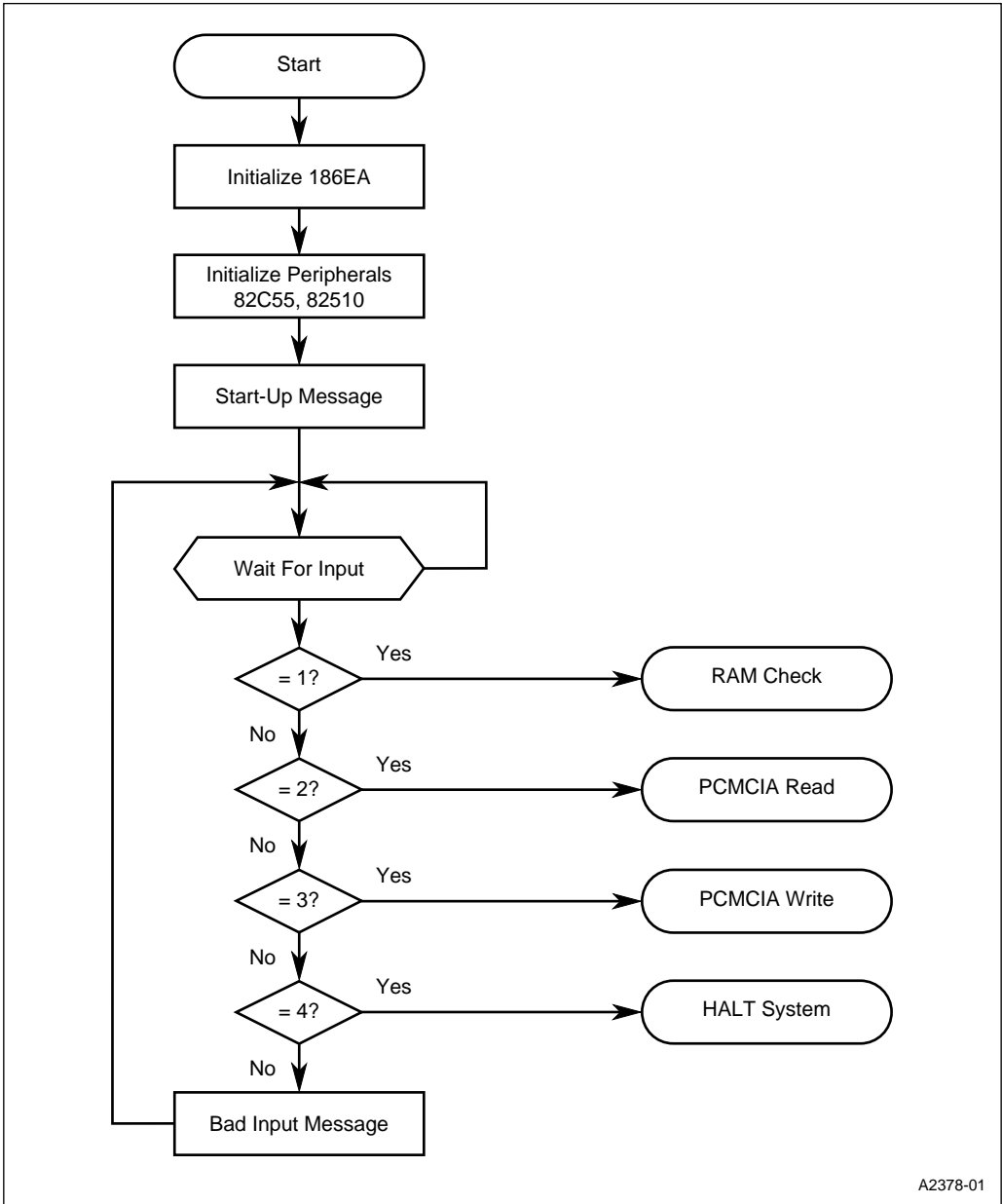
The 82510 serial controller is initialized for polled operating mode as described in application note AP-401 [7]. To address the 64 Mbyte address space, the 80C186EA system performs the following steps:

- Sets up the appropriate page address using the 82C55. Each page is 64 Kbytes in length, and the page can be set up by writing to the port.
- Performs the memory read, write, or erase operation in the PCMCIA page address space of the 80C186EA (physical address 50000H).
- Rewrites the page address space of the 82C55 to indicate Page 0.

The above three-step procedure has appropriate messages to indicate the actions taken. The messages are printed to the screen using the 82510 serial controller. However, the task of writing, reading, or erasing the Flash memory card is accomplished by the special memory management software. The flowcharts are shown in Figures 5–8. Also, the system can detect the density and speed of the Series 2 Flash memory cards.

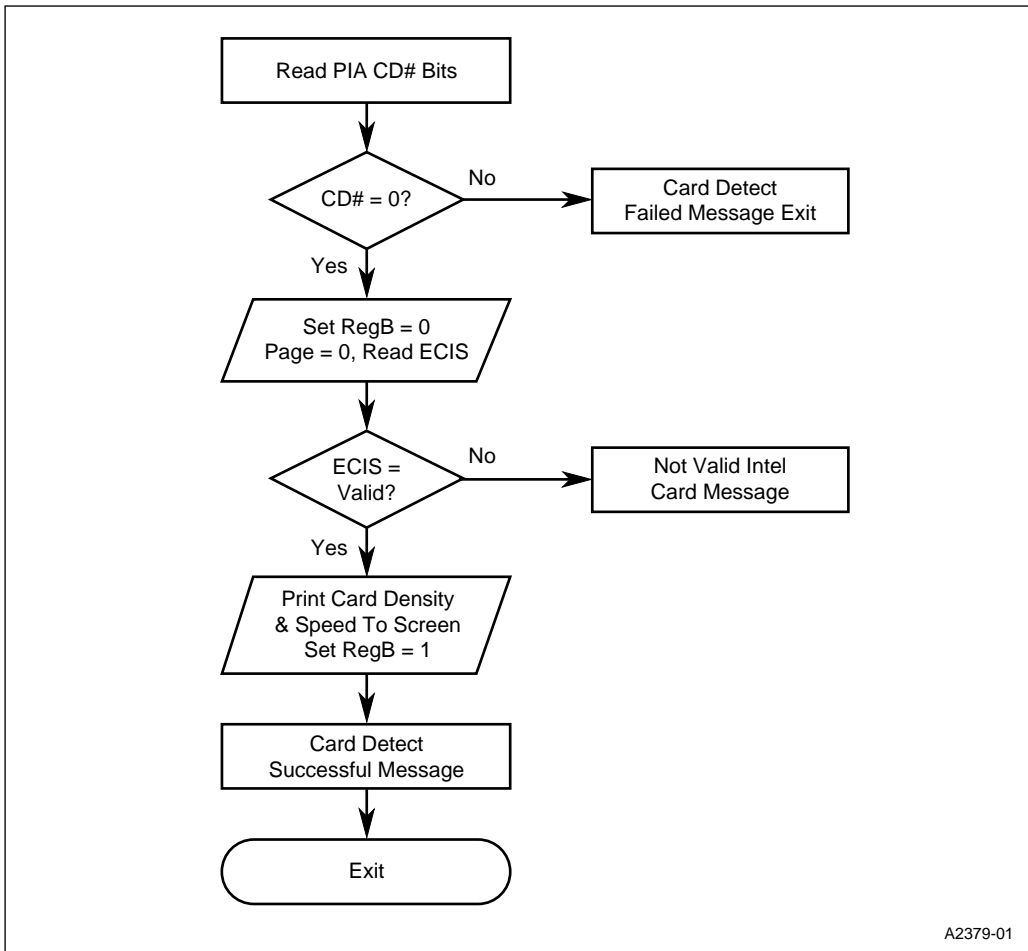
Another advantage of adding Flash memory cards, as mentioned earlier, is that they provide the capability of developing and debugging code remotely and then executing it on an existing platform or system. Also, because Flash memory cards are memory-mapped, it is possible to add new features or change the existing software very easily, either by using the card's memory or by downloading to the system memory. Code can be executed from the PCMCIA card in several ways such as FAR JUMPS, FAR CALLS or RETurn instructions [8].





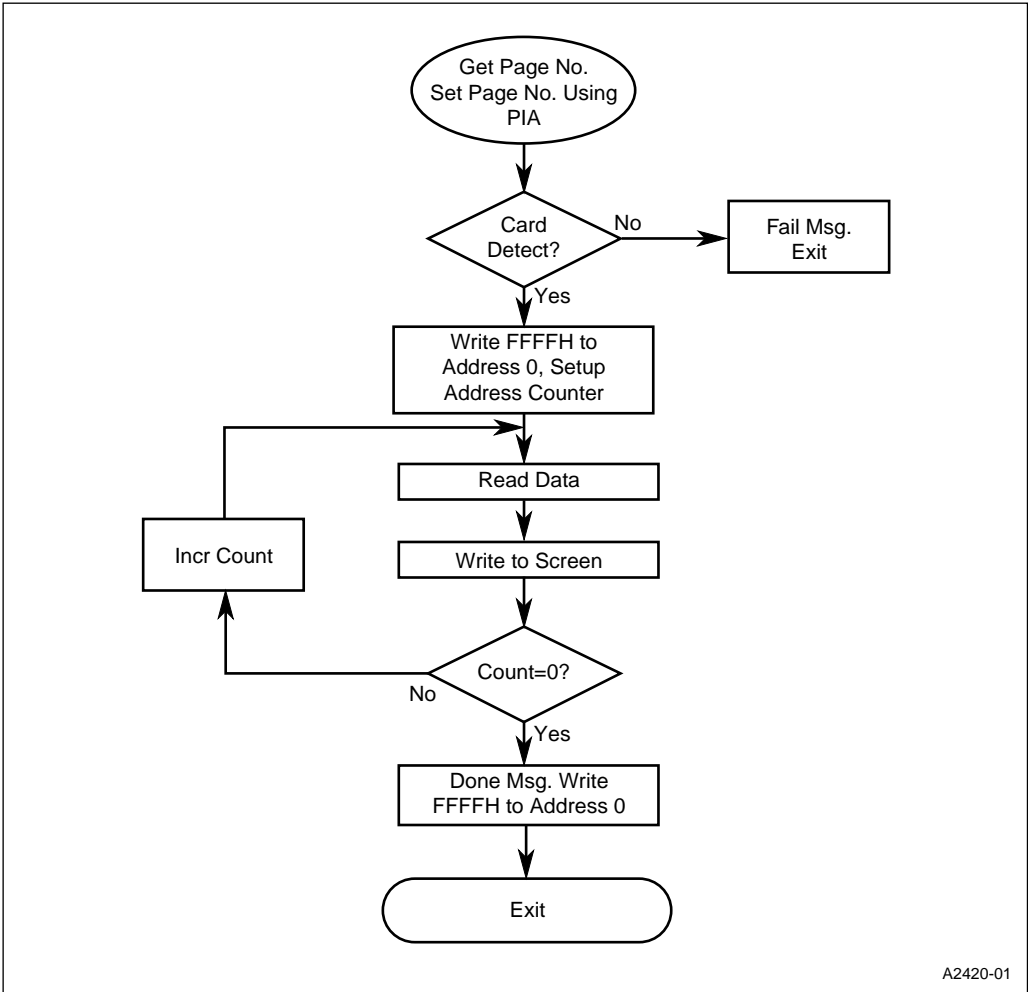
A2378-01

Figure 4. System Initialization Flowchart



**Figure 5. Card-Detect Flowchart**

- FAR JUMPs can be used to achieve intrasegment jumps in code. This instruction uses no stack space. The syntax for this instruction is `JUMP FAR LABEL your_label`, where *your\_label* is the user's chosen label. However, this allows no direct way to get back to the original location.
- FAR CALLs are just like regular procedure calls, except that they allow intrasegment operation. This instruction uses 4 bytes of stack space for the current Code Segment (CS) and Instruction Pointer (IP) values. The syntax for this instruction is `CALL PROC (FAR) or CALL DWORD PTR[reg16]`. In this case, a RET instruction at the end will take you back to the previous code segment, if necessary, as the information of the previous CS and IP are stored on the stack.

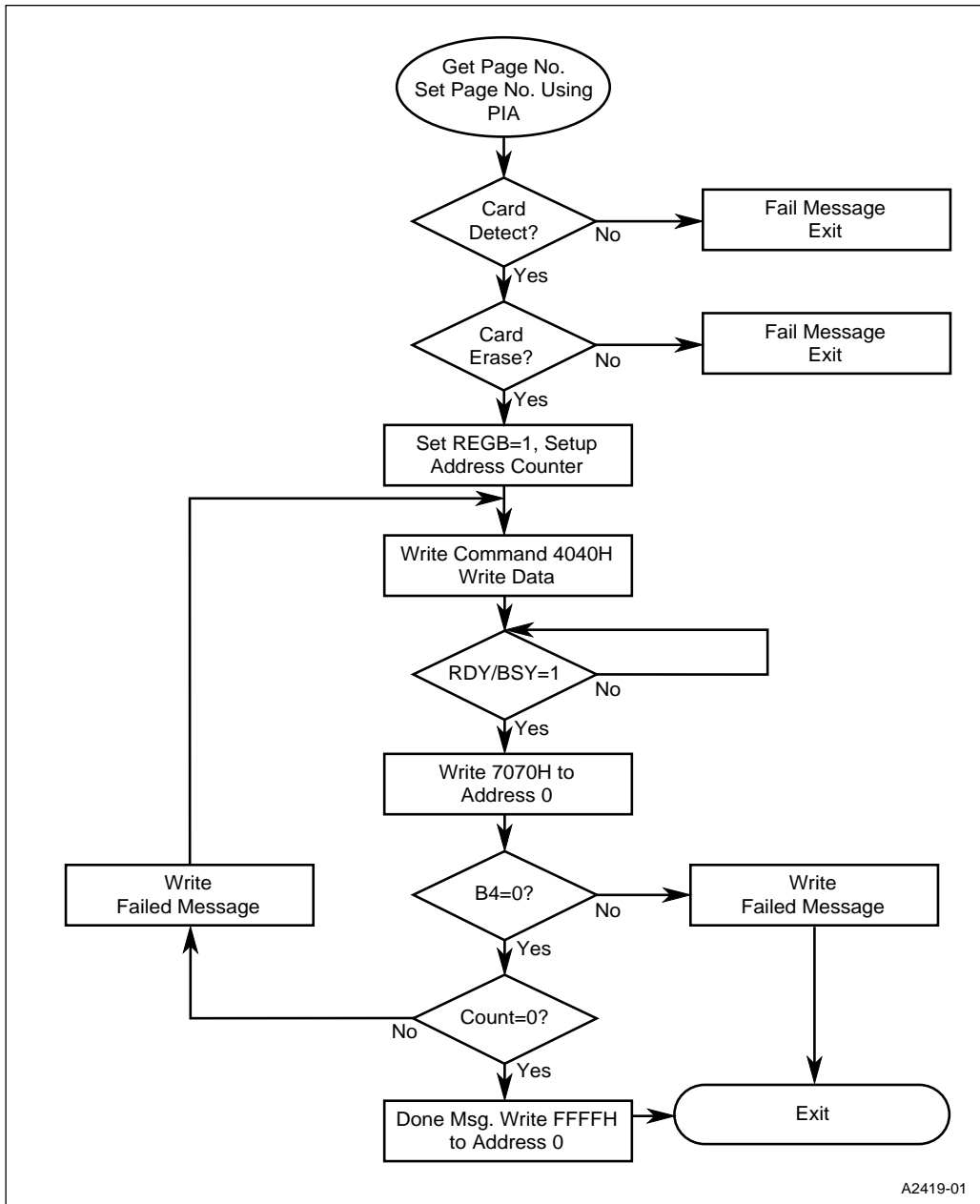


**Figure 6. Read from PCMCIA Card Flowchart**

- Using the RET instruction uses the stack only temporarily, to achieve the desired jump. The desired CS and IP are first stored on the stack, then the RET instruction restores the current CS and IP. The syntax for this instruction is RET.

The RET instruction was used in this application. First the new IP and CS, which correspond to the PCMCIA card page value (i.e., CS=5000H and IP=0000H), were saved on the stack, then a RET instruction was executed. This

method, as mentioned before, temporarily uses 4 bytes of stack space. Now the code can be executed like any other segment with relocatable code present on the memory card. Again the correct page has to be set up before the RET instruction executes. At the end of the code segment, another RET can be executed to get to the main program. The overhead of a few instructions compared to the 64 Kbyte segment of code is minimal and the performance degradation is also negligible.



A2419-01

Figure 7. Write to PCMCIA Card Flowchart

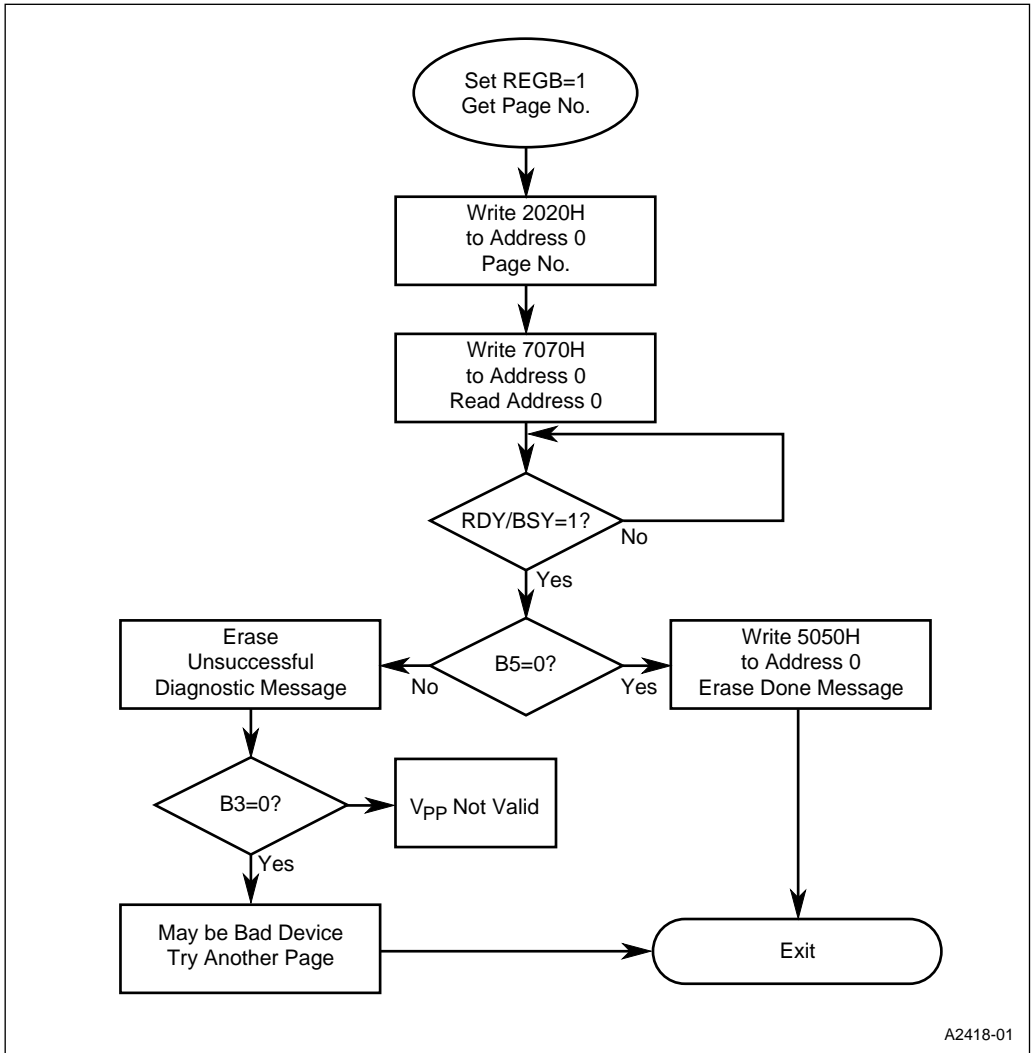


Figure 8. PCMCIA Card Erase Flowchart

A2418-01

## 6.0 CONCLUSION

This article describes an 80C186EA-PCMCIA interface design and outlines the advantages of such a design. Software files and drivers can be obtained by calling Intel's applications BBS at 916-356-3600 in the U.S. or at +44(0) 793-496340 in Europe.

## REFERENCES

[1]. James D. Broesch, "Standard DOS Tools to Generate ROMable 80x86 code," *EDN*, Jan 7, 1993.

[2]. *Embedded Microcontrollers and Processors, Vol II*, pp. 24-59 thru 24-173. Intel, 1993, Lit. # 210830.

[3]. Larry Bates, "Quick Upgrade from the 80C186 to the 80C186EA," AP-468, Intel, 1991, Lit.# 272157.

[4]. *Memory Products, Series 2 Flash Memory Cards*, Intel, 1993, Lit. # 290434.

[5]. *PC Card Interface Controller*, Intel, 1993, Lit# 290423.

[6]. *Peripherals Components*, pp 3-124 thru 3-146, Intel, 1993, Lit # 231256.

[7]. Faisal Imdad-Haque, "Designing with the 82510 Asynchronous Serial Controller," AP-401, Intel, 1993, Lit. # 231928.

[8]. *iAPX 86, 88, 186 and 188 User's Manual, Programmer's Reference*, Intel, 1986, Lit.# 210911.



# APPENDIX A

## MONITOR AND CONTROL PROGRAM FOR 186EA-PCMCIA INTERFACE

```
;PCMCIA-186 INTERFACE PROJECT TEST CODE
; MEMORY MAP:
; CHIP          START   STOP     CS#
; 27C64         FE000   FFFFF   UCS#
; 84256         00000   0FFFF   LCS#
; 28F256        60000   6FFFF   MCS2# 1 wait state
; PCMCIA        50000   5FFFF   MCS1# Page starts here
; 82510         0000   PCS0#   IO MAP
; 82C55         0080   PCS1#   IO MAP
; PCBREGISTER   IN IO MAP @ FF00H
;=====
;
;Code written in ASSEMBLY ASM86 Intel.
;program invoked by "ASM86 AP186.ASM LIST"
;*****
;Set control block register addresses
CNTRL_BLK      EQU      0FF00H; Relocation of PCB at reset
UMCS           EQU      0FFA0H; Upper chip select reg location
LMCS           EQU      0FFA2H; Lower chip select reg location
MMCS           EQU      0FFA6H; Middle chip select reg locations
MPCS           EQU      0FFA8H; Control register for PCS & MCS
PACS           EQU      0FFA4H; Peripheral chip select reg
PWRSAV         EQU      0FFF0H; Power save register
POWRCON        EQU      0FFF2H; Power control register
STEPID         EQU      0FFF6H; Device step ID
RELREG         EQU      0FFFEH; Relocation register
;Base address definition
PCMCIA_BASE    EQU      5000H ; PCMCIA card page start
PCS_BASE       EQU      0000 ; Base address
CODE_BASE      EQU      0FC00H; Code starts here

;Other registers
PCS0           EQU      PCS_BASE; PCS0# select base address
PCS1           EQU      PCS0+128; Address for PCS1#
PCS2           EQU      PCS0+256; Base address for PCS2#
;*****
;Control Register Values And Port Addresses
;Remember the ports are aligned at even addresses
;(A1(OF 186)-> to A0 of parallel port)
;*****
CLRSCRN        EQU      1AH ;Clear screen for VT100
CNTR_PIO       EQU      PCS1+6 ;82C55 Control register
PIO_A          EQU      PCS1 ;Port A
PIO_B          EQU      PCS1+2 ;Port B
PIO_C          EQU      PCS1+4 ;Port C
SIO_BAUD_LO    EQU      PCS0 ;Baud low register (DLAB=1)
SIO_GER        EQU      PCS0+2 ;General purpose enable register(BANK=0)
```

```

SIO_GIR      EQU      PCS0+4      ;Interrupt control/bank register(BANK=0)
SIO_LCR      EQU      PCS0+6      ;Line configuration register(BANK=0)
SIO_DATA     EQU      PCS0        ;Data port for RXD and TXD (BANK=1)
SIO_GSR      EQU      PCS0+14     ;General status register (BANK=1)
SIO_ICM      EQU      PCS0+14     ;Internal command register (BANK=1)
SIO_FMD      EQU      PCS0+2      ;TxFIFO=0, RxFIFO=0 (BANK=2)
SIO_TMD      EQU      PCS0+6      ;Modem automatic (BANK=2)
SIO_RMD      EQU      PCS0+14     ;Rx Mc mode register (BANK=2)
SIO_IMD      EQU      PCS0+8      ;General status register(BANK=2)
SIO_RIE      EQU      PCS0+12     ;Rx Intr enable reg (BANK=2)
SIO_CLCF     EQU      PCS0        ;Clock configuration register (BANK=3)
SIO_BACF     EQU      PCS0+2      ;BRGA config reg
SIO_BBCF     EQU      PCS0+6      ;BRGB config reg
;*****
;
;Definition of register values and constants
;1 Wait state for memory
;3 (max) Wait states for IO
;*****
UMCS_VAL     EQU      0FC38H      ;16 KByte blocks
LMCS_VAL     EQU      0FFCH       ;64 KByte blocks
MPCS_VAL     EQU      0A0BFH      ;64 KByte MCSn chip-select blocks
MMCS_VAL     EQU      41FDH       ;Start address of MCSn BLOCK=20000H
PACS_VAL     EQU      003FH       ;Start address of PCSn BLOCK=0000H

;OTHER CONTROL VALUES
PIO_VAL      EQU      89H         ;Set up for 82C55
SIO_IMD_VAL  EQU      0CH         ;IAM=1, RFD=1
SIO_LCRD_VAL EQU      83H         ;DLAB=1
SIO_LCR_VAL  EQU      03H         ;DLAB=0, 8 BIT 16X, NP, 4/4 MODE
SIO_RMD_VAL  EQU      10H         ;SAMPL WINDOW=7/16
SIO_CLCF_VAL EQU      50H         ;16x MODE, RxCS=TxCS=BRGA
SIO_BAL_VAL  EQU      3CH         ;DLAB=1: 9600 BAUD, (BAH=0 DEFAULT)
SIO_GER_VAL  EQU      07H         ;Tx disable, Rx enable intr
SIO_RIE_VAL  EQU      0          ;Disable all functions
SIO_GIR1_VAL EQU      21H         ;Bank 1
SIO_GIR2_VAL EQU      41H         ;Bank 2
SIO_GIR3_VAL EQU      61H         ;Bank 3
SIO_BACF_VAL EQU      04         ;BRGA mode
SIO_BBCF_VAL EQU      04         ;BRG mode
SIO_ICM_VAL  EQU      10H         ;Software reset

;CONSTANTS
CR           EQU      0DH          ;Carriage return
LF           EQU      0AH          ;Line feed
EOT          EQU      04H          ;End of text, ASCII stopper
EOL          EQU      0A0DH        ;LF, CR
TOS          EQU      0FFFFH       ;Top of stack at top of RAM
RAM_BASE     EQU      1024         ;Usable RAM starts here
RAMTOP_USE   EQU      TOS-50H     ;Usable RAMTOP area
RAM_COUNT    EQU      RAMTOP_USE-RAM_BASE ;Actual RAM count
INMSG        EQU      RAM_BASE+1   ;Save input value
CMDSTAT      EQU      00H
PAGE_STORE   EQU      410H
PAGE_NUM_VAL EQU      PAGE_STORE+40H
PAGE_NUMBR   EQU      PAGE_NUM_VAL+10

```



```
PAGE_SIZE      EQU      PAGE_NUMBR+4
CD_SP_VAL      EQU      PAGE_STORE+2
CD_SZ_VAL      EQU      CD_SP_VAL+1
```

CODE SEGMENT AT 0FC00H

ASSUME CS:CODE, DS:CODE

INIT:

```
;*****
;
;      Entry Point On Power Up
;
;*****
```

```
FW_START LABEL FAR      ;Forces far jump
      CLI                ;Disable Interrupts
      MOV AX, 0           ;Initialize data segment to 00000
      MOV DS, AX         ;DS=00000
      MOV ES, AX         ;ES=00000
      MOV SS, AX         ;SS=00000
      MOV AX, TOS        ;SP=FFFFH
      MOV SP, AX
```

```
-----
;Initialization of stack is done; STACK=0FFFFH
;Set up chip selects
;
;UCS - EPROM Select (Initialized during POWER_ON code)
;LCS - SRAM Select (Set to SRAM Size)
;PCS - I/O Select (PCS0-1 Support 82C55 AND 82510)
;MCS0 -FLASH Select (Set to 64 Kbyte Size)
;MCS1 PCMCIA Select (Set to 64 Kbyte Size)
-----
```

```
      MOV DX, LMCS        ;Set up LCS Register
      MOV AX, LMCS_VAL
      OUT DX, AL          ;Remember, byte writes OK

      MOV DX, MPCS        ;Ready for PCS lines 0-1
      MOV AX, MPCS_VAL    ;As well as MCS programming
      OUT DX, AL

      MOV DX, MMCS        ;Set up PCMCIA-FLASH chip-select
      MOV AX, MMCS_VAL
      OUT DX, AL

      MOV DX, PACS        ;Set up I/O chip-select
      MOV AX, PACS_VAL
      OUT DX, AL
```

```
;*****
;FILL_A and B interrupt vectors point to unwanted interrupt so that
;stray interrupts won't cause the board to hang
;Verify that DS=00000H
;*****
      MOV DI, 0           ;Start at 0
      MOV CX, 256        ;Do 256 times
```

```

FILL_A:
    MOV     [DI], OFFSET UNWANTED_INT
    ADD     DI, 4                ;Fill offsets
    LOOP   FILL_A              ;IP=Offset

    MOV     DI, 2                ;Start at 2
    MOV     CX, 256             ;Do 256 times
    MOV     AX, CODE_BASE       ;Equate CS=0FC00H

FILL_B:
    MOV     [DI], AX
    ADD     DI, 4                ;Fill CS segments
    LOOP   FILL_B

;*****
; Set Up PCMCIA-186 INTERRUPT VECTORS
;
;Set up Vector 6 to point to routine that catches the execution of
;unknown opcodes. This routine is especially important when debugging
;new code since it is possible that the new code can vector off into
;never-never land.
;*****
    MOV     DI, 4*6              ;Point to Vector 6 (unused Opcode)
    MOV     [DI], OFFSET OPCODE_TRAP ;Load offset
    MOV     [DI+2], AX          ;Load segment

;*****
;Initialize parallel I/O
;82C55 is at $80, connects to PCS1, no interrupts, direct
;polling
;*****
    CALL   PIO_INIT             ;Call parallel I/O initialize

;*****
;Initialize Serial I/O
;82510 is at $0, connects to PCS0, receive and
;transmit are polled.
;*****
    CALL   SIO_INIT             ;Call Initialize Serial IO

;*****
; Output a Menu to the Screen
;*****
    STI                     ;Enable all interrupts
SCRN_MENU:
    EVEN
    MOV     AX, TOS           ;Re-initialize the SP
    MOV     SP, AX
    MOV     AX, CODE_BASE    ;Re-initialize the DS
    MOV     DS, AX           ;register. DS=CS
    LEA     BX, MENU_MSG     ;Get bad input message
    CALL   CARD_PRINT        ;Print to the screen
;*****
; What is the Keyboard Input?
;*****
    MOV     BX, INMSG
    CALL   RECEIVE           ;Get the character

```

```

MOV     [BX], AL           ;Check the input value
CALL    TRANSMIT          ;Echo back the character
CMP     AL, '1'           ;Is it one?
JZ      RAMTEST           ;If so, it is RAM test
CMP     AL, '2'           ;Is it two?
JZ      RD_PCMCIA         ;Read from PCMCIA card
CMP     AL, '3'           ;Is it 3?
JZ      WR_PCMCIA         ;Write to PCMCIA card
CMP     AL, '4'           ;Get out?
JZ      LOOP_HALT        ;Halt
LEA     BX, BAD_INPUT_MSG ;Get bad input message
CALL    CARD_PRINT        ;Print the message to screen
CALL    DELAY             ;Wait for some time
JMP     SCR_NMENU         ;Go back and output a message to screen
LOOP_HALT:
LEA     BX, HALT_MSG      ;Load halt message to the pointer
CALL    CARD_PRINT        ;Print to screen
CALL    DELAY             ;Wait for some time
HLT     ;System halted

;*****
; Checkerboard RAM Test Routine
;*****
RAMTEST:
EVEN
CALL    RAM_TEST          ;Test the locations
CALL    DELAY             ;Wait for some time
JMP     SCR_NMENU         ;Go back to the menu

;*****
; PCMCIA Read Routine
;*****
RD_PCMCIA:
EVEN
CALL    CARD_DETECT       ;Detect card presence?
CALL    PAGE_INPUT        ;Get the page #'s
CALL    RDPCMCI           ;Read from PCMCIA card
JMP     SCR_NMENU         ;Go back to screen menu

;*****
; PCMCIA Write Routine
;*****
WR_PCMCIA:
EVEN
CALL    CARD_DETECT       ;Valid card?
CALL    PAGE_INPUT        ;Get the page #'s
CALL    CARD_ERASE        ;Erase PCMCIA card
CALL    WRPCMCI           ;Write to PCMCIA card
JMP     SCR_NMENU         ;Go back to screen menu

;*****
; A simple delay routine provides time to read messages
; (uses CX)
;*****
DELAY:
EVEN

```

```

    PUSH    AX                ;Save registers
    PUSH    CX
    PUSH    DS
    XOR     AX, AX            ;Clear AX
    XOR     CX, CX            ;Clear CX
    MOV     DS, AX            ;Clear DS
LOOP_DELAY:
    DEC     CX                ;Decrement CX
    NOP
    NOP
    CMP     CX, AX            ;Is CX=AX?
    JNZ     LOOP_DELAY        ;If not, loop back
    POP     DS
    POP     CX                ;Restore values
    POP     AX
    RET

;*****
; This subroutine gets the page number, modifies the input,
; and prints an appropriate message.
; Calls GET_PAGE
; Converts the input ASCII value to hex
; Page # in hex is less than 400H.
; Returns the value in AX.
; Calls CARD_PRINT
;*****
    EVEN
PAGE_INPUT:
    PUSH    AX
    PUSH    DS                ;Save registers
    XOR     AX, AX
    MOV     DS, AX
    CALL    NEXT_LINE         ;Print a blank line
    LEA     BX, PG_NUM_MSG    ;Output first message
    CALL    CARD_PRINT        ;to screen
    MOV     BX, PAGE_NUM_VAL  ;Store page numbers
    CALL    GET_PAGE          ;Get multiple characters
    MOV     BX, PAGE_NUMBR    ;Get the actual value
    MOV     AX, [BX]          ;to set up page
    CMP     AX, 'MR'          ;Is it bad page input?
    JZ     OUT_BAD            ;Get out
    MOV     DX, PIO_A         ;Port A
    OUT     DX, AL
    MOV     DX, PIO_B         ;Port B
    IN     AL, DX
    AND     AL, 80H           ;Save REGB# input
    OR     AL, AH             ;Get high byte
    OUT     DX, AL            ;Set up Port B
    POP     DS
    POP     AX
    RET
OUT_BAD:
    POP     DS
    POP     AX                ;Get out of this subroutine and
    JMP     SCRNMENU          ;to the main routine
;*****

```

```

; Subroutine GET_PAGE
; Gets the values from keyboard and echos back
; Converts ASCII to hex
; *****
GET_PAGE:
    PUSH    AX            ;Save accumulator
    PUSH    CX            ;Save count reg
    MOV     CX, 0         ;Start count
PGNUMLP:
    CALL    RECEIVE       ;Get character
    MOV     [BX], AL      ;Save the value
    CALL    TRANSMIT     ;Transmit the value
    INC     BX            ;Increment pointer
    INC     CX            ;Increment count
    CMP     AL, CR        ;Is it the end?
    JNZ    PGNUMLP       ;If not, loop back
    MOV     AL, EOT       ;Store control char
    MOV     [BX], AL
    MOV     BX, PAGE_NUM_VAL;Start conversion
    MOV     AX, 0H        ;Clear AX
    MOV     DX, 0         ;Clear DX
    CMP     CX, 04
    JNZ    BAD_PAGE      ;If not, page is bad
    ADD     BX, 2         ;Add 2
    CALL    ASC_HEX
    MOV     DL, AL        ;Move it to DL
    DEC     BX
    CALL    ASC_HEX
    SHL    AL, 4         ;Shift left four times
    MOV     DH, AL        ;
    DEC     BX
    CALL    ASC_HEX
    MOV     AH, AL        ;Get the high byte
    MOV     AL, 0
    OR     AL, DH         ;Get second nibble
    OR     AL, DL         ;Make it one word
    MOV     BX, PAGE_NUMBR ;Get page number place
    MOV     [BX], AX      ;Store the value
    MOV     BX, PAGE_SIZE ;Get the card size value
    MOV     CX, [BX]      ;Get the value of CX
    CMP     AX, CX        ;Compare the values
    JG     BAD_PAGE
    JMP     GET_IP_END    ;Go to end
BAD_PAGE:
    MOV     AX, 'MR'      ;If bad put sign
    MOV     BX, PAGE_NUMBR ;Get page number place
    MOV     [BX], AX      ;Store sign
    LEA    BX, BAD_PG_MSG ;Bad page message
    CALL    CARD_PRINT    ;Print the message
    JMP     SCR�N_MENU
GET_IP_END:
    POP     CX
    POP     AX            ;Restore regs
    RET     ;Return
; *****
; ASC_HEX converts the received characters to hex

```

```

;*****
ASC_HEX:
    MOV     AL, [BX]           ;Get the data
    SUB     AL, '0'           ;Take out 30H
    CMP     AL, 09H          ;Is it between 0-9, inclusive?
    JG      ASC_LP
    JMP     ASC_END          ;Else, go to end
ASC_LP: SUB     AL, 07         ;Else, subtract 7
ASC_END:RET
;*****
; Transmit a character using 82510
; Nothing returned
; Nothing modified
;*****

    EVEN
TRANSMIT:
    PUSH   CX                 ;Save CX register
    PUSH   DS
    XOR    CX, CX             ;Clear CX register
    MOV    DS, CX             ;Load ES to point to RISM memory
    MOV    CX, AX             ;Save copy of AX
TR_ST_LOOP:
    MOV    DX, SIO_GSR        ;Check transmit buffer
    IN     AL, DX             ;Get the status data
    TEST   AL, 10H            ;Save TxM status
    JE     TR_ST_LOOP         ;If not, loop
    MOV    AX, CX             ;Copy back value
    MOV    DX, SIO_DATA       ;Setup to output to 82510 data port
    OUT    DX, AL             ;Write the data to the 82510
    POP    DS
    POP    CX                 ;Restore stack
    RET                       ;Return home
;*****
;Subroutine to Print Next Line
;*****
NEXT_LINE:
    MOV    AX, EOL            ;Move EOL data into AX
    CALL   TRANSMIT           ;Send out the data in AL
    MOV    AL, AH             ;Copy AH into AL
    CALL   TRANSMIT           ;Send out the data in AL
    RET
;*****
; Receive a character using 82510
; Returns:
; Nothing modified
;*****

    EVEN
RECEIVE:
    PUSH   DS
    XOR    AX, AX             ;Clear AX
    MOV    DS, AX             ;DS=0000H
RX_ST_LOOP:

```

```

MOV     DX, SIO_GSR           ;Check receive buff
IN      AL, DX                ;Get the status data
TEST    AL, 04H               ;Save RxM status
JNE     RX_ST1_LOOP          ;If not, loop
TEST    AL, 01H               ;Save RxM status
JE      RX_ST_LOOP           ;If not, loop
RX_ST1_LOOP:
MOV     DX, SIO_DATA          ;Setup to output to 510 data port
IN      AL, DX                ;Get the data
POP     DS                    ;Restore stack
RET                                           ;Return home
;*****
; Checkerboard RAM Test
;*****
RAM_TEST:
EVEN
PUSH    AX                    ;Save contents
PUSH    BX
PUSH    CX
PUSH    DS
XOR     AX, AX                ;AX=0
MOV     DS, AX                ;Reload data segment
MOV     AX, 5555H              ;Load the checkerboard pattern
LOOP2:
MOV     DX, AX                ;Make a copy in DX
MOV     BX, RAM_BASE+1        ;Load start address into BX
MOV     CX, RAM_COUNT         ;Load count value into CX
CLD
RT_LP:
MOV     [BX], AX              ;Write data to the RAM
MOV     AX, [BX]              ;Copy back
CMP     AX, DX                ;Compare AX to DX
JE      EQUAL                 ;If equal, jump
JMP     BAD_RAM               ;If not, go to bad RAM message
EQUAL:
INC     BX
LOOPNE  RT_LP                 ;Else loop
CLC                                           ;Clear carry
RCL     AX, 1                  ;Repeat checkerboard test
CMP     AX, 5554H              ;Is it done with two loops?
JE      DONE                   ;If so, go to done
JMP     LOOP2                  ;Else, loop back
DONE:
LEA     BX, RAM_OK_MSG         ;Output RAM OK message
CALL    CARD_PRINT             ;to screen
POP     DS
POP     CX
POP     BX
POP     AX
RET

BAD_RAM:
;*****
; Output "RAM bad message" to the screen,
; call card-print and halt the system
;*****

```

```

LEA    BX, BAD_RAM_MSG ;Output message
CALL  CARD_PRINT      ;Output characters to screen
HLT

;*****
; Card-detect Routine
; Checks to see whether if a valid card is present
; If card is detected, Print Intel series 2 message to screen
; Message will be modified to present speed and density info.
;*****
EVEN
CARD_DETECT:
PUSH  AX              ;Save accumulator
PUSH  DS              ;Save data seg
XOR   AX, AX         ;Load accum with zero
MOV   DS, AX         ;DS points to zero
CALL  NEXT_LINE      ;Go to next line to print
MOV   DX, PIO_C      ;Set up for Port C of PIA
IN    AL, DX         ;Get the data
AND   AL, 11000000B  ;Save CD1, CD2
CMP   AL, 80H        ;Are they one?
JNZ   CFLMSG         ;If not zero, go to card fail
MOV   DX, PIO_B      ;Point to Port B
MOV   AL, 0
OUT   DX, AL         ;Reg#=0
MOV   DX, PIO_A      ;Point to Port A
OUT   DX, AL         ;Page=0
MOV   AX, PCMCIA_BASE ;Load base address of PCMCIA card
MOV   ES, AX         ;Load extra segment with base
MOV   CX, 35         ;Load count value
MOV   SI, 00H        ;Load source pointer
MOV   DI, PAGE_STORE ;Load destination address
STD
CD_LOOP1:
MOV   AX, ES:[SI]    ;Get the data
MOV   DS:[DI], AL    ;Save low byte
ADD   SI, 2
INC   DI
LOOP  CD_LOOP1       ;Finish loop
MOV   AX, EOL        ;Save EOL
MOV   DS:[DI], AX
ADD   DI, 2
MOV   AL, EOT
MOV   DS:[DI], AL    ;Load control characters
MOV   BX, CD_SZ_VAL  ;Get data
CALL  CARD_SIZE      ;Get card density
MOV   BX, CD_SP_VAL  ;Get data
CALL  CARD_SPEED     ;Get card speed
MOV   BX, PAGE_STORE ;Get data
ADD   BX, 21         ;Card manufacturer msg
MOV   AL, [BX]       ;Get next character
CD_TR: CALL TRANSMIT  ;Transmit characters
INC   BX             ;Increment pointer
MOV   AL, [BX]       ;Get next character
CMP   AL, EOT        ;Is it the end?

```



```

    JNZ     CD_TR           ;Transmit loop
    POP     DS             ;Get back DS
    JMP     CD_END        ;End of card detect

CFLMSG:
    POP     DS             ;Save data seg.
    LEA    BX, MSG_FAIL   ;Load message to indicate PCMCIA
    MOV    AL, [BX]       ;Card not present
CFLTR: CALL    TRANSMIT   ;Transmit characters
    INC    BX             ;Increment pointer
    MOV    AL, [BX]       ;Get next character
    CMP    AL, EOT        ;Is it the end?
    JNZ    CFLTR         ;Transmit loop
    JMP    SCR_N_MENU     ;Do the message

CD_END:
    MOV    DX, PIO_B     ;Point to Port B
    MOV    AL, 80H       ;
    OUT   DX, AL         ;Reg#=1
    POP    AX
    RET

;*****
; Compare the card speed and output proper message
;*****
CARD_SPEED:
    PUSH   DS
    MOV    AL, [BX]      ;Card present?
    MOV    AH, AL        ;Save a copy
    CMP    AL, 52H       ;Is it 200ns?
    JZ     CARD_T0NS
    MOV    AL, AH        ;Copy back
    CMP    AL, 51H       ;Is it 250ns?
    JZ     CARD_TFNS
    MOV    AL, AH        ;Copy back
    CMP    AL, 53H       ;Is it 150ns?
    JNZ    CARD_UNKNWN   ;Else, output card unknown message
    LEA    BX, CARD_150  ;Load card message
    CALL   CARD_PRINT    ;Print the message
    JMP    CD_SPEND      ;End of card detect

CARD_T0NS:
    LEA    BX, CARD_200  ;Load card message
    CALL   CARD_PRINT    ;Print card message
    JMP    CD_SPEND      ;End of card detect

CARD_TFNS:
    LEA    BX, CARD_250  ;Load card message
    CALL   CARD_PRINT    ;Print card message
    JMP    CD_SPEND      ;End of card detect

CARD_UNKNWN:
    LEA    BX, UNKNWN_MSG ;
    CALL   CARD_PRINT    ;
    JMP    SCR_N_MENU

CD_SPEND:
    POP    DS
    RET

;*****
; Card-size Check Routine
;*****

```

```

CARD_SIZE:
  PUSH    DS                ;Save segment
  MOV     AL, [BX]         ;Get data into AL
  MOV     AH, AL           ;Copy AL
  MOV     BX, PAGE_SIZE    ;Store value at pointer
  CMP     AL, 06H         ;Is it 2M?
  JE      CD_2MEG
  CMP     AL, 0EH         ;Is it 4M?
  JE      CD_4MEG
  CMP     AL, 16H         ;Is it 6M?
  JE      CD_6MEG
  CMP     AL, 1EH         ;Is it 8M?
  JE      CD_8MEG
  CMP     AL, 26H         ;Is it 10M?
  JE      CD_10MEG
  CMP     AL, 2EH         ;Is it 12M?
  JE      CD_12MEG
  CMP     AL, 3EH         ;Is it 16M?
  JE      CD_16MEG
  CMP     AL, 4EH         ;Is it 20M?
  JE      cd_20MEG

CD_UNKNSZ:
  LEA    BX, CARD_MISZ    ;Get mis-size msg
  CALL   CARD_PRINT      ;Print the message
  JMP    SCRNMENU

CD_2MEG:
  MOV    AX, 10H*2-1      ;# of pages in 2M minus 1
  MOV    [BX], AX
  LEA    BX, CARD_2MEG    ;Get 2M message
  JE     SIZE_END

CD_4MEG:
  MOV    AX, 10H*4-1      ;# of pages in 4M minus 1
  MOV    [BX], AX
  LEA    BX, CARD_4MEG    ;Get 4M message
  JE     SIZE_END

CD_6MEG:
  MOV    AX, 10H*6-1      ;# of pages in 6M minus 1
  MOV    [BX], AX
  LEA    BX, CARD_6MEG    ;Get 6M message
  JE     SIZE_END

CD_8MEG:
  MOV    AX, 10H*8-1      ;# of pages in 8M minus 1
  MOV    [BX], AX
  LEA    BX, CARD_8MEG    ;Get 8M message
  JE     SIZE_END

CD_10MEG:
  MOV    AX, 10H*10-1     ;# of pages in 10M minus 1
  MOV    [BX], AX
  LEA    BX, CARD_10MEG   ;Get 10M message
  JE     SIZE_END

CD_12MEG:
  MOV    AX, 10H*12-1     ;# of pages in 12M minus 1
  MOV    [BX], AX
  LEA    BX, CARD_12MEG   ;Get 12M message

```

```

        JE          SIZE_END
CD_16MEG:
        MOV        AX, 10H*16-1      ;# of pages in 16M minus 1
        MOV        [BX], AX
        LEA        BX, CARD_16MEG    ;Get 16M message
        JE          SIZE_END
CD_20MEG:
        MOV        AX, 10H*20-1      ;# of pages in 20M minus 1
        MOV        [BX], AX
        LEA        BX, CARD_20MEG    ;Get 20M message
        JE          SIZE_END

SIZE_END:
        CALL       CARD_PRINT        ;Print the message
        POP        DS
        RET

;*****
; Print the Card Message Routine
;*****
CARD_PRINT:
        PUSH       DS
        MOV        AX, CODE_BASE      ;CS SEG address
        MOV        DS, AX             ;Get DS straight
        MOV        AL, [BX]           ;Get next character
PR_LP1: CALL       TRANSMIT           ;Transmit characters
        INC        BX                 ;Increment pointer
        MOV        AL, [BX]           ;Get next character
        CMP        AL, EOT            ;Is it the end?
        JNZ        PR_LP1             ;Transmit loop
        CALL       NEXT_LINE          ;
        POP        DS
        RET

;*****
; Card-erase Routine
;*****
EVEN
CARD_ERASE:
        PUSH       DS
        PUSH       AX
        MOV        DX, PIO_B
        MOV        AL, 80H
        OUT        DX, AL             ;Reg#=1
        MOV        AX, PCMCIA_BASE    ;Get the base address
        MOV        DS, AX             ;Reset DS
        MOV        AX, 2020H          ;Set up for erase
        MOV        DI, 0              ;Set up destination pointer
        MOV        [DI], AX           ;Start erase
        MOV        AX, 0D0D0H         ;Erase confirm
        MOV        [DI], AX           ;
        MOV        CX, 0              ;Get a copy reg ready
        MOV        DX, PIO_C          ;Get Port C address
ER_LP1: IN         AL, DX              ;Get Port C value
        MOV        AH, AL             ;Make a copy
        AND        AL, 00001000B      ;RDY/BSY#=1?

```

```

CMP     AL, 08H
JNZ     ER_LP1      ;Keep looping
MOV     AX, [DI]    ;Get it to AX
AND     AX, 2020H   ;Erase done?
CMP     AX, 2020H   ;
JZ      NO_ERASE   ;If so, erase is not done
MOV     AX, 0FFFFH  ;Check status
MOV     [DI], AX    ;Reg of PCMCIA card
MOV     AX, 5050H   ;Get the cntrl word
MOV     [DI], AX    ;Get it out
LEA     BX, ER_SUC_MSG ;Load erase done message
CALL    CARD_PRINT  ;Print it to screen
POP     AX
POP     DS
RET

NO_ERASE:
LEA     BX, ER_UN_MSG ;Load unsuccessful message
CALL    CARD_PRINT  ;Print it to screen
POP     AX
POP     DS
JMP     SCR_N_MENU  ;Go back to start

;*****
;This routine reads out of the PCMCIA card.
;Now it just send out a card-not-present message.
;*****
RDPCMCIA:
PUSH    DS          ;Save data segment
PUSH    AX          ;Save accumulator
MOV     DX, PIO_B   ;Set up reg#=1
IN      AL, DX      ;Get the present value
MOV     AH, 80H     ;
OR      AL, AH      ;
OUT     DX, AL      ;Output the modified value
MOV     AX, PCMCIA_BASE ;Load PCMCIA base
MOV     DS, AX      ;Load DS
MOV     AX, 0FFFFH  ;
MOV     SI, 0       ;Reset source pointer
MOV     [SI], AX    ;Ready the card for read
MOV     CX, 3FFH    ;Start the count
RD_LP1: MOV     AX, [SI] ;Start reading
CALL    TRANSMIT    ;Put it to screen
MOV     AL, AH      ;Get high byte
CALL    TRANSMIT    ;
ADD     SI, 2       ;Add to pointer
LOOP   RD_LP1      ;Get back
CALL    NEXT_LINE
LEA     BX, RD_MSG  ;Read success
CALL    CARD_PRINT  ;Print this to screen
MOV     AX, 0FFFFH  ;
MOV     SI, 0       ;Reset source pointer
MOV     [SI], AX    ;Ready the card for next read
POP     AX
POP     DS
RET

;*****

```

```

;This routine writes from the PCMCIA card.
;Now it just send out a card not present message.
;*****
WRPCMCIA:
    PUSH    DS            ;Save data segment
    PUSH    AX            ;Save accumulator
    MOV     DX, PIO_B     ;Set up reg#=1
    IN     AL, DX         ;Get the present value
    MOV     AH, 80H       ;
    OR     AL, AH         ;
    OUT    DX, AL         ;Output the modified value
    MOV     AX, PCMCIA_BASE ;Load ES with destination
    MOV     DS, AX
    MOV     AX, CODE_BASE ;Get the source pointer
    MOV     ES, AX
    LEA    SI, MENU_MSG   ;Load the source value
    MOV     DI, 0         ;Start address of destination
    MOV     BX, 0
    MOV     AX, 0FFFFH
    MOV     [BX], AX
    MOV     CX, MSG_COUNT ;Load the count value
WR_LP1: MOV     AX, 4040H   ;Set up for write
    MOV     [BX], AX      ;Write command
    MOV     AX, ES:[SI]   ;Get the data
    MOV     DS:[DI], AX   ;Save word
    ADD     SI, 2
    ADD     DI, 2         ;Next location
    MOV     DX, PIO_C     ;Get Port C input
WR_LP0: IN     AL, DX      ;Is RDY/BSY#=1?
    AND    AL, 8
    CMP    AL, 08H
    JNZ    WR_LP0        ;If not, loop
    MOV    AX, 7070H     ;Read status reg
    MOV    [BX], AX      ;Write
    MOV    AX, [BX]      ;Read status
    AND    AX, 1010H     ;Write status?
    CMP    AX, 1010H     ;Has it written properly
    JZ     NO_WRITE     ;Else, no write
    LOOP  WR_LP1        ;Finish loop
    MOV    AX, 0FFFFH    ;
    MOV    [BX], AX      ;Ready for read
    LEA    BX, WR_DONE_MSG ;
    CALL   CARD_PRINT    ;Done message
    POP    AX
    POP    DX
    RET
NO_WRITE:
    POP    AX
    POP    DS
    LEA    BX, NO_WR_MSG ;Unsuccessful message
    CALL   CARD_PRINT
    JMP    SCR_N_MENU    ;Go to screen menu
;*****
;This procedure initializes the parallel I/O.
;82C55 set up in mode 0

```

```

;Port A, B output ports (latched)
;Port C input port (not latched)
;*****
PIO_INIT:
    PUSH DS                ;Save DS
    XOR AX, AX             ;Clear AX register
    MOV DS, AX             ;Re-initialize DS
    MOV DX, CNTR_PIO       ;Load pointer
    MOV AX, PIO_VAL        ;82C55 set up complete
    OUT DX, AL
    MOV AL, 0              ;Output 0'S to the ports
    MOV DX, PIO_A          ;Load Port A to pointer
    OUT DX, AL             ;Port A, Port B are outputs
    ADD DX, 2
    OUT DX, AL
    POP DS
    RET
;*****
; This procedure initializes the serial I/O.
; 82510 set-up in polled mode for
; receive operation only
; No buffer space for Rx and Tx
; Baud rate=9600
; N 8 1 mode
; Initialize 82510
;*****
SIO_INIT:
    PUSH DS                ;Save DS
    XOR AX, AX             ;Re-initilize DS
    MOV DS, AX
    MOV DX, SIO_GIR        ;Switch to bank 1
    MOV AX, SIO_GIR1_VAL   ;Prepare for reset
    OUT DX, AL
    MOV DX, SIO_ICM
    MOV AL, SIO_ICM_VAL    ;Do a software reset
    OUT DX, AL
    MOV DX, SIO_LCR        ;Set up communication parameters
    MOV AL, SIO_LCRD_VAL   ;No parity, 1 stop, 8-bit char
    OUT DX, AL
    MOV DX, SIO_BAUD_LO    ;Set up bal register
    MOV AL, 3CH            ;Low part of 9600 baud
    OUT DX, AL
    MOV DX, SIO_LCR        ;DLAB=0
    MOV AL, SIO_LCR_VAL    ;8 Data bits
    OUT DX, AL
    MOV DX, SIO_GIR        ;Switch to bank 2
    MOV AL, SIO_GIR2_VAL
    OUT DX, AL
    MOV DX, SIO_FMD        ;TxFIFO=0, RxFIFO=0
    MOV AL, 0
    OUT DX, AL
    MOV AL, 18H
    MOV DX, SIO_TMD        ;Automatic mode for modem
    OUT DX, AL
    MOV DX, SIO_IMD
    MOV AL, SIO_IMD_VAL    ;IAM=1, RFD=1

```

```

OUT     DX, AL
MOV     DX, SIO_RIE
MOV     AL, SIO_RIE_VAL           ;Disable all functions
OUT     DX, AL
MOV     DX, SIO_RMD
MOV     AL, SIO_RMD_VAL         ;Sample window=7/16th
OUT     DX, AL
MOV     DX, SIO_GIR             ;Switch to bank 3
MOV     AL, SIO_GIR3_VAL
OUT     DX, AL
MOV     DX, SIO_CLCF           ;Set up clock configuration reg
MOV     AL, SIO_CLCF_VAL       ;16X mode, BRGA Output
OUT     DX, AL
MOV     DX, SIO_BACF           ;Set up BRGA configuration reg
MOV     AL, SIO_BACF_VAL       ;BRGA mode
OUT     DX, AL
MOV     DX, SIO_BBCF           ;Set up BRGB configuration reg
MOV     AL, SIO_BBCF_VAL       ;BRG mode
OUT     DX, AL
MOV     DX, SIO_GIR             ;Switch to bank 0
MOV     AL, 01
OUT     DX, AL
MOV     DX, SIO_GER             ;Enable Rx, Tx interrupts
MOV     AL, SIO_GER_VAL        ;INT0 interrupts are now enabled
OUT     DX, AL
MOV     AL, SIO_GIR1_VAL
MOV     DX, SIO_GIR             ;Switching to bank 1
OUT     DX, AL
POP     DS
RET                                     ;Return from initialization

;*****
;Unwanted interrupts have occurred
;output a simple message and halt
;*****
UNWANTED_INT:
    PUSH    AX                    ;Save used registers
    PUSH    DX                    ;Screw up in code
    LEA     BX, BAD_INT_MSG       ;Output message
    MOV     AL, [BX]              ;Get characters
BINT_TRLP:
    MOV     DX, SIO_DATA          ;Set up to output to 510 data port
    OUT     DX, AL                ;Write the data to the 510
    INC     BX                    ;Increment pointer
    MOV     AL, [BX]              ;Get next character
    TEST    AL, EOT               ;Is it the end?
    JNZ    BINT_TRLP             ;Transmit loop
    HLT

;*****
;Opcode is wrong, just halt.
;*****
EVEN
OPCODE_TRAP:
    LEA     BX, BAD_TRAP_MSG      ;Output message
    MOV     AL, [BX]              ;Get characters
BTRP_TRLP:

```

```

MOV     DX, SIO_DATA      ;Set up to output to 510 data port
OUT     DX, AL            ;Write the data to the 510
INC     BX                ;Increment pointer
MOV     AL, [BX]         ;Get next character
TEST    AL, EOT          ;Is it the end?
JNZ     BTRP_TRLP        ;Transmit loop
HLT

;*****
; General Message Area
;*****

EVEN
BAD_INPUT_MSG DB CR,LF,'KEY BOARD INPUT IS BAD, PLEASE USE PROPER
CHOICE',CR,LF,EOT
MENU_MSG      DB LF,CR
DB '*****',LF,CR
DB 'WELCOME TO PCMCIA-186 INTERFACE',LF,CR
DB ' TO CHECK SYSTEM FUNCTIONALITY,',LF,CR
DB ' ENTER ANY OF THE FOLLOWING CHOICE.',LF,CR,LF,CR
DB ' 1. FOR RAM TEST',CR,LF
DB ' 2. FOR READING FROM PCMCIA CARD',CR,LF
DB ' 3. FOR WRITING TO PCMCIA CARD',CR,LF
DB ' 4. EXIT', CR, LF
DB '*****',CR,LF
DB ' > ',EOT
RAM_OK_MSG    DB CR,LF,'RAM TESTED OK. GOOD JOB ! ',CR,LF,EOT
BAD_RAM_MSG   DB CR,LF,'RAM TESTED BAD, RESET SYSTEM! GOOD LUCK ! ',LF,CR,EOT
BAD_INT_MSG   DB CR,LF,'UN-WANTED INTERRUPTS HAPPENED, SYTEM HALTED',LF,CR,EOT
BAD_TRAP_MSG  DB CR,LF,'CAN NOT DECIPHER THE OPCODE, SYTEM HALTED!',LF,CR,EOT
HALT_MSG      DB CR,LF,'SYSTEM HALTED AS PER YOUR REQUEST!',EOT
MSG_FAIL      DB CR,LF,'CARD DETECT FAILED.',CR,LF,EOT
CARD_150      DB LF,'CARD SPEED=150ns',EOT
CARD_200      DB LF,'CARD SPEED=200ns',EOT
CARD_250      DB LF,'CARD SPEED=250ns',EOT
CARD_2MEG     DB LF,'CARD SIZE=2Meg Bytes',EOT
CARD_4MEG     DB LF,'CARD SIZE=4Meg Bytes',EOT
CARD_6MEG     DB LF,'CARD SIZE=6Meg Bytes',EOT
CARD_8MEG     DB LF,'CARD SIZE=8Meg Bytes',EOT
CARD_10MEG    DB LF,'CARD SIZE=10Meg Bytes',EOT
CARD_12MEG    DB LF,'CARD SIZE=12Meg Bytes',EOT
CARD_16MEG    DB LF,'CARD SIZE=16Meg Bytes',EOT
CARD_20MEG    DB LF,'CARD SIZE=20Meg Bytes',EOT
UNKNWN_MSG    DB LF,CR,'SPEED UNKNOWN, PLEASE CHECK, MAY BE NON-INTEL CARD',LF,CR
DB LF,CR,'OR SERIES 1 CARD OR BY DIFFERENT MANUFACTURER',LF,CR,EOT
CARD_MISZ     DB LF,CR,'SIZE UNKNOWN. PLEASE CHECK',EOT
PG_NUM_MSG    DB LF,CR,'PLEASE ENTER THE PAGE NUMBER',CR,LF
DB LF,CR,'FOR THE ABOVE CARD SIZE IN HEX',CR,LF
DB LF,CR,'FOLLOWED BY <CR>',LF,CR
DB LF,CR,'1 MEG = "00F" PAGES AND SO ON..',CR,LF
DB LF,CR,'UP TO 64 MEG OR "3FF" PAGE',CR,LF,EOT
BAD_PG_MSG    DB LF,CR,'ENTERED PAGE NUMBER IS INCORRECT',CR,LF
DB LF,CR,'REFER TO THE INSTRUCTION AND',LF,CR
DB LF,CR,'RE-TYPE THE VALUE, please',LF,CR,EOT
RD_MSG        DB LF,CR,'CARD READ IS SUCCESSFUL',LF,CR
DB LF,CR,'WELL DONE, THANK YOU!',LF,CR,EOT
ER_SUC_MSG    DB LF,CR,'SELECTED PAGE WAS SUCCESSFULLY ERASED',LF,CR

```



```

                DB LF,CR,'GREAT JOB, WELL DONE',LF,CR,EOT
ER_UN_MSG      DB LF,CR,'SELECTED PAGE CAN NOT BE ERASED',LF,CR
                DB LF,CR,'PLEASE TRY ANOTHER PAGE OR CHECK THE Vpp',CR,LF,EOT
WR_DONE_MSG    DB CR,LF,'SELECTED PAGE WRITTEN SUCCESSFULLY',LF,CR
                DB CR,LF,'GREAT JOB, WELL DONE',CR,LF,EOT
NO_WR_MSG      DB LF,CR,'CAN NOT WRITE TO THE CARD',LF,CR
                DB LF,CR,'CHECK THE CARD FUNCTIONALITY',CR,LF,EOT
MSG_COUNT      EQU      $-MENU_MSG
CODE           ENDS
;*****
;      Power-on Reset Code to Get Started
;      No wait state for memory, block size=8 KBytes
;*****
        ASSUME CS:POWER_ON

        POWER_ON SEGMENT AT 0FFFFH

PRST:     MOV  DX, UMCS           ;Point to UMCS Register
          MOV  AX, UMCS_VAL      ;Reprogram UMCS to match

          OUT  DX, AL            ;system requirements
          JMP  FW_START          ;Jump to init code

POWER_ON   ENDS
;*****
;
;      Data Segment
;
;*****
DATA      SEGMENT PUBLIC  'DATA'
          DD   256 DUP (?)      ;Reserved for Interrupt Vectors

;The definitions of all card codes follow

DATA      ENDS
;*****
; Program Ends
;*****

        END  INIT

```

