Old Microprocessor Circuit Evaluation

John Smigel

25 March 2022

Abstract

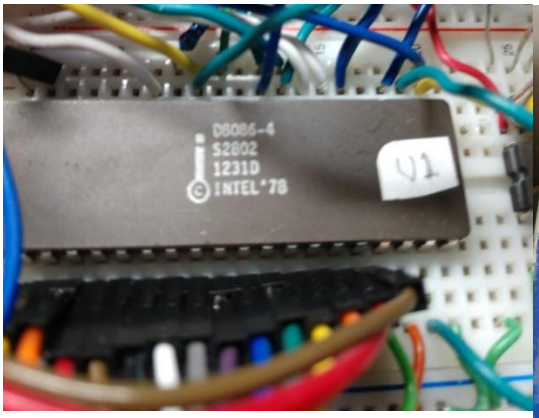
This report analyzes an old Intel MCS-86 Microcomputer System University Kit that I bought in 1978 and never finished building. My goal has been to determine what components are still working and develop the minimum system capable of input and output to a remote terminal (or keypad and LED displays). The analysis uses components from the kit connected with a prototyping breadboard on a cookie sheet.

The kit's microprocessor is a 16-bit 4MHz 8086-4 preliminary release. The 8086 was released after the 8-bit 8080 and before the more-popular 8-bit 8088 and 8085 processors. The kit also included most of the chips needed for a 'min' system (RAM, address latches, serial port chip, and interrupt controller chip), min and max system design examples, PROMs with monitor code, monitor code listing, and device spec sheets booklet.

The key components, 8086-4 processor and 8284 clock chip, still worked. The RAM and PROM memory chips no longer worked; I had to rewrite the monitor code to test the system. I didn't test the interrupt controller chip, so I don't know if it still worked. In addition to the RAM and PROMs not working, the serial interface chip also did not work, but I may have broken that during testing. I replaced the no-longer-working or broken components with similar obsolete, slightly newer components.

The breadboard prototyping system was adequate for construction and evaluation, but was prone to connections coming out or loose. Probably it was still easier than construction using soldered or wire-wrap connections, as I originally planned.

The main test equipment included: a Rigol DS1054 digital 4-channel 1G S/S oscilloscope, a Kingst LA2016 logic analyzer, and an Atmel STK600 microcontroller development system with ATMEGA 1284P microcontroller. Development software included Microchip Studio, NASM assembler, Octave (free MATLAB), and the Kingst VIS logic analyzer software.



1. ASSUMPTIONS

I made the following assumptions associated with this analysis:

a) Some part of the original kit would still work and I could find replacements for broken parts

b) The prototyping breadboard was adequate for this type of circuit at 4 MHz

c) Only relatively inexpensive test equipment would be required

d) Someone might be interested in the results of this analysis

Fortunately, the key components worked and I did not have to replace them. At times I replaced the 8086-4 with a 'compatible' HM-8086 processor to see if the processor was the problem. The HM-8086 worked better in some ways (cleaner outputs) and worse in others. I did not use the nominal 5MHz clock when I swapped in the HM-8086, so that might have been a problem. Most of the components or similar substitutes can be found for sale somewhere. I had bought replacements and additional components (memory, bus controller, etc.) a long time ago. It is difficult to find DIP40 (40-pin dual inline package) chips now. The major electronics sellers don't carry or are phasing them out. I used to buy from Digikey, but they now have a minimum quantity for everything of about $300-$400 per item and don't carry DIP40. I switched to using Mouser electronics. I spent more on shipping than on the few items I needed. I could not initially find a replacement uart chip in DIP40 format, so I bought a PLCC44 chip and an adapter meant for another chip. I modified the adapter to work with the uart and it worked out ok. I did buy a DIP40 uart chip from China on eBay for a few dollars, but did not end up needing it.

At times I wished I had soldered everything together. I have the components to produce a soldered or wire-wrapped circuit board. I might try to assemble it that way for fun. The breadboard wires become a mess and can be easy to pull out and hard to get in the correct hole.

The most expensive test equipment needed was the Rigol oscilloscope at about $419 (current price – I paid a little less when I bought it a few years ago). If all the components worked, you could probably get away with just using the logic analyzer(LA). I first tried the cheapest analyzer you can buy, about $10. It works with free, open system, Pulseview software that is pretty good. I had a lot of problems with the cheap LA (not reliable and could not be used with some connections) so I had to upgrade to a better one.

The last assumption, d, is probably a bad one.

2. RESULTS

Figure 2-1 shows the output of the 8086 test code on a terminal emulator (PuTTY). The code outputs a text string and waits for input at a dot prompt. If a command of G, S, or D is entered, it acknowledges the command by outputting G, D, or S Received. If any other key is pressed, an error routine outputs an asterisk.

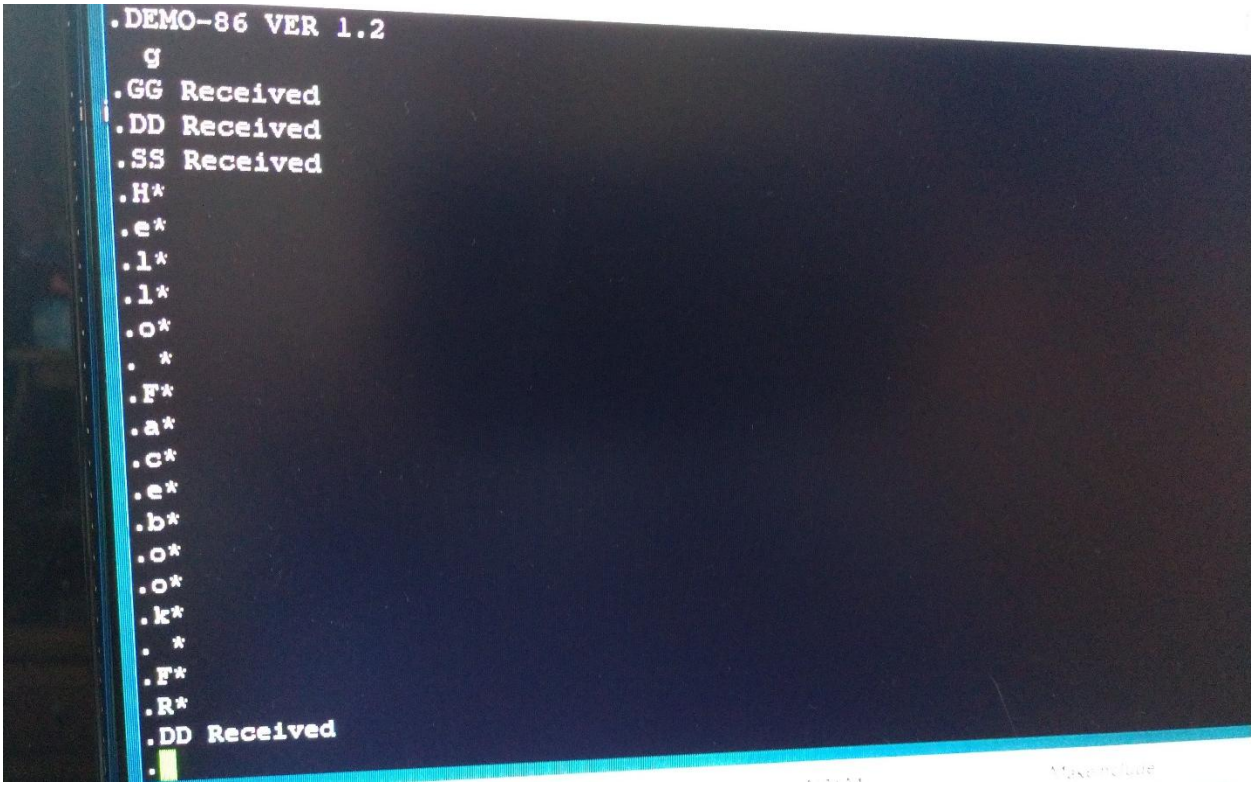


Figure 2-1. Example 8086 Processor Output Shows It Works!

It took about 2 months to get this extremely useful circuit and app together and working. I first checked the programmable read-only memory (PROM) and random access memory (RAM) chips because they looked in the worst physical shape and tend to break. None of the PROM or RAM chips appeared to work. I used test equipment and software I developed using a SDK600 microcontroller system. I don't know for sure that the chips don't work because my test software could have been the problem. I could go back and test them with my latest software, but don't want to bother.

Since the PROM chips didn't work, I needed to develop a way to get machine code into the processor. I developed an electronically erasable PROM (EEPROM) programmer using the SDK-600 and replaced the original Ultraviolet-erasable PROMs (UVPROMs) with EEPROMs from Microchip. The really old RAM chips also did not appear to work, so I replaced them with some old RAM chips I had bought a long time ago in case I built my own PC. It might be hard to find

DIP40 RAM chips now, but I have a lot of them. The replacement RAM and ROM chips have more memory than the originals, but I am only using 4k bytes on each chip, with 2 chips used together to support a 16-bit word.

The next hurdle was getting the serial interface chip to work. I tried to get the original chip from the kit to work, but could not get it to produce any output. I was able to communicate with the front end of the chip and download data and commands, but never could get output. I might have broken it because I accidentally connected the output directly to an RS-232 (original design) interface cable without the required receiver/driver chips (I had the chips too). I didn't want to use the chips/RS-232 because they required +- 12 or 15 volt power. I ended up buying a universal asynchronous receiver/transmitter (UART)-to-universal serial bus(USB) cable to bypass using the RS-232 of olden days. I could not find a DIP40 replacement so used a PLCC44 chip and modified an adapter to get all the connections I needed. The 44 pin chip only output 40 of the 44 signals to the 40-pin adapter. Wouldn't you know it; I needed 2 of the 4 missing connections. I had to solder wires to the adapter to get the missing connections.

After testing the new UART by itself, I started testing everything together. I found a lot of wiring mistakes and a few broken logic devices. I rewired the main bus to use multicolor ribbon cable. Eventually I decided the hardware was mostly working and switched to fixing the monitor code. Most of the code problems had to do with accessing the segmented memory properly. There are 4 segment registers, stack (SS), code (CS), data (DS), and extra (ES). The processor uses a 20-bit address that is produced by combining 4 bits of the segment register with an address offset. You can view my notes for details on what it took to get everything working.

3. DISCUSSION

3.1 Method of Attack

A prototyping breadboard and relatively modern test equipment have been used to evaluate the survival of 44-year-old electronics components in an old university evaluation kit.

3.2 Effect of Assumptions on Results

The solderless breadboard proved adequate to determine that the processor still worked. The test equipment required was the most expensive part of the analysis. The oscilloscope is the most expensive, but has amazing capability for its cost.

3.3 Evaluation of Results

This analysis shows that some types of digital logic chips can last a long time. It depends on how they are constructed and stored. The particular conductive foam that the kit chips were stored in was later found to corrode the pins. The chips from the kit that survived the 44-year storage were ceramic 40-pin DIP chips. I was pleasantly surprised that the 8086-4 processor chip worked since it was a preliminary release of the chip. I only implemented the min system (so far). I had originally had grand plans to build a max system that could actually be used for something. Now your watch or TV remote has more processing power and memory.

I have several newer 8086's or compatible's that I could check if they still work too. They generally have plastic cases and run at higher clock frequencies. The 4 versions of the chip are: 1) 8086-4 (4 MHz), 8086 (5 MHz), 8086-1 (10 MHz), and 8086-2 (8 MHz). The input clock or crystal for the chips is three times the CPU frequency. I'm not sure if my clock chip will work at all frequencies or just 4 (12 MHz). I have a slightly newer clock chip that might work.

3.4 Suggestions for Future Effort

I am open to suggestions for what to try next. I could try a max system. Or developing enough so that a bare-bones operation system could be installed (Linux Mint? would be difficult). I have one other started, but unfinished project to develop a custom field-programmable gata array (FPGA)-based circuit board to interface an original IBM-PC video output with a more-modern RGB monitor.

Other possible projects include: analysis of solar cells output and coherence, or developing a smart phone/tablet app for helping to keep score in games such as Phase 10 or Pay Me.

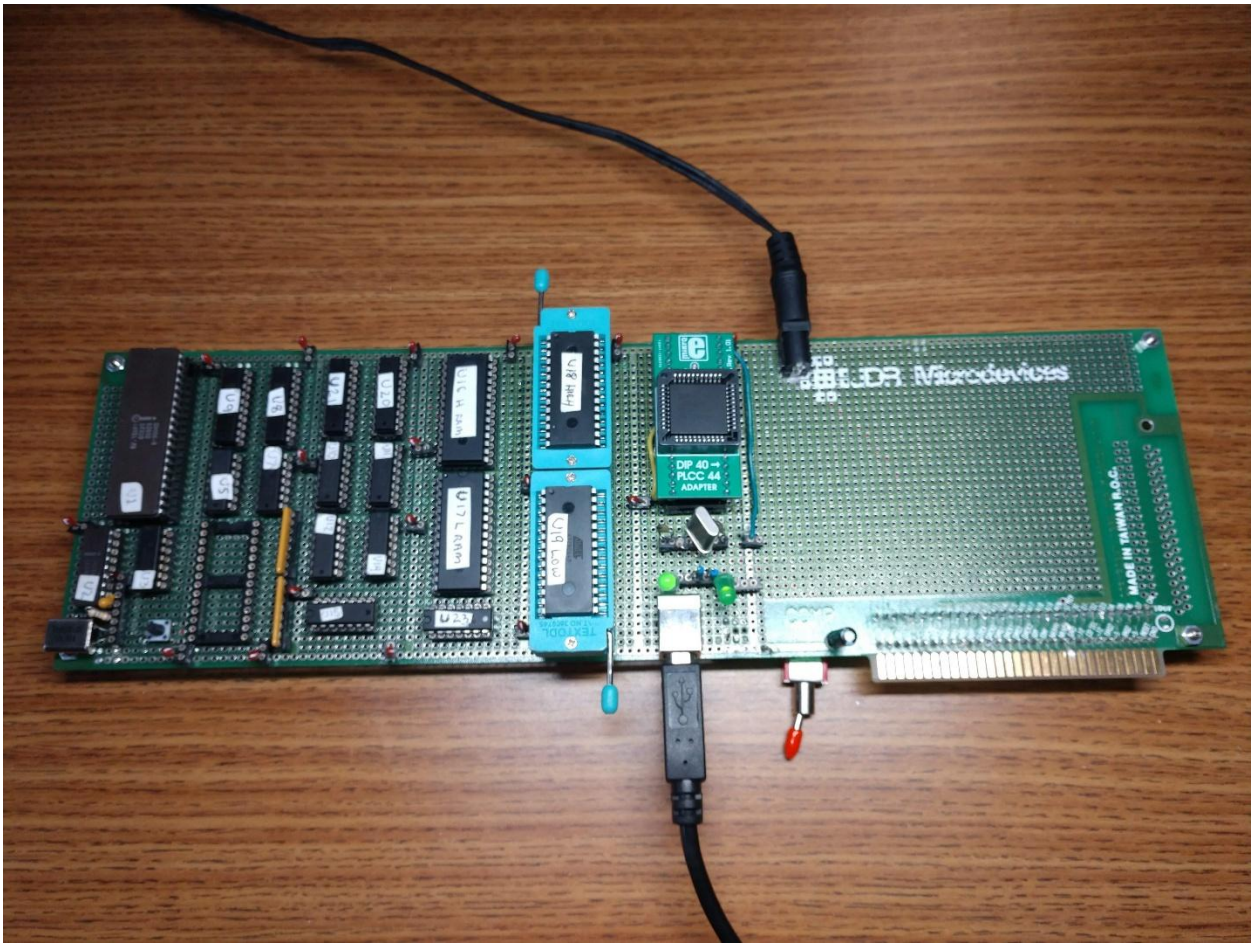
Anything anyone would be interested in seeing??

4. ANALYSIS

4.1 System Design Summary

The completed hardware is shown in Figure 4-1. So many wires you can't see the logic chips.

Version 1



Version 2

Figure 4-1. *Crazy Computer Mess*

The final circuit design is shown in Figures 4-2 and 4-3. A parts list and program listings are shown in Appendices A to D.

The empty socket in version 2 is for an 8259A interrupt controller. I was not able to get an interrupt controller chip to work completely. I could get interrupts, but the processor was unable to read the correct interrupt number. I'm not sure if this was a controller chip, processor chip, circuit design, or interrupt signal problem. May investigate further in the future.

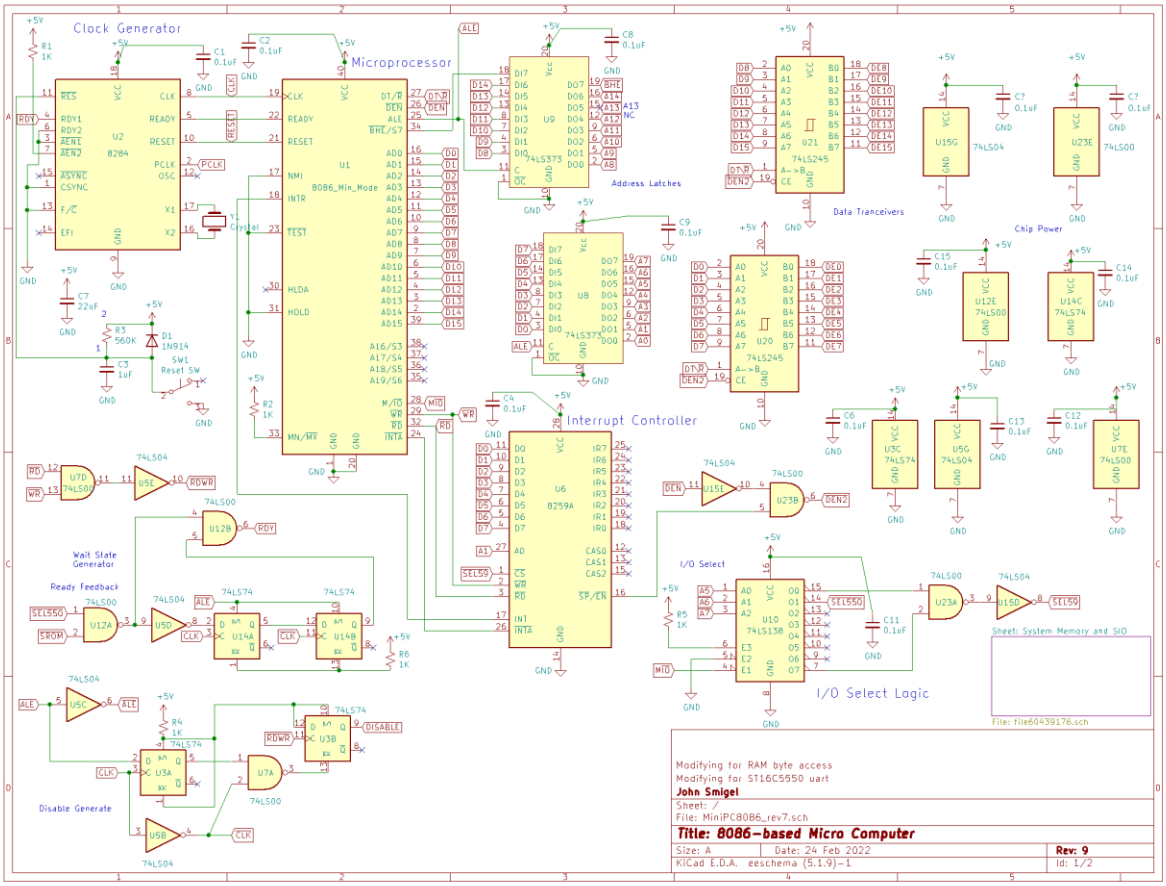
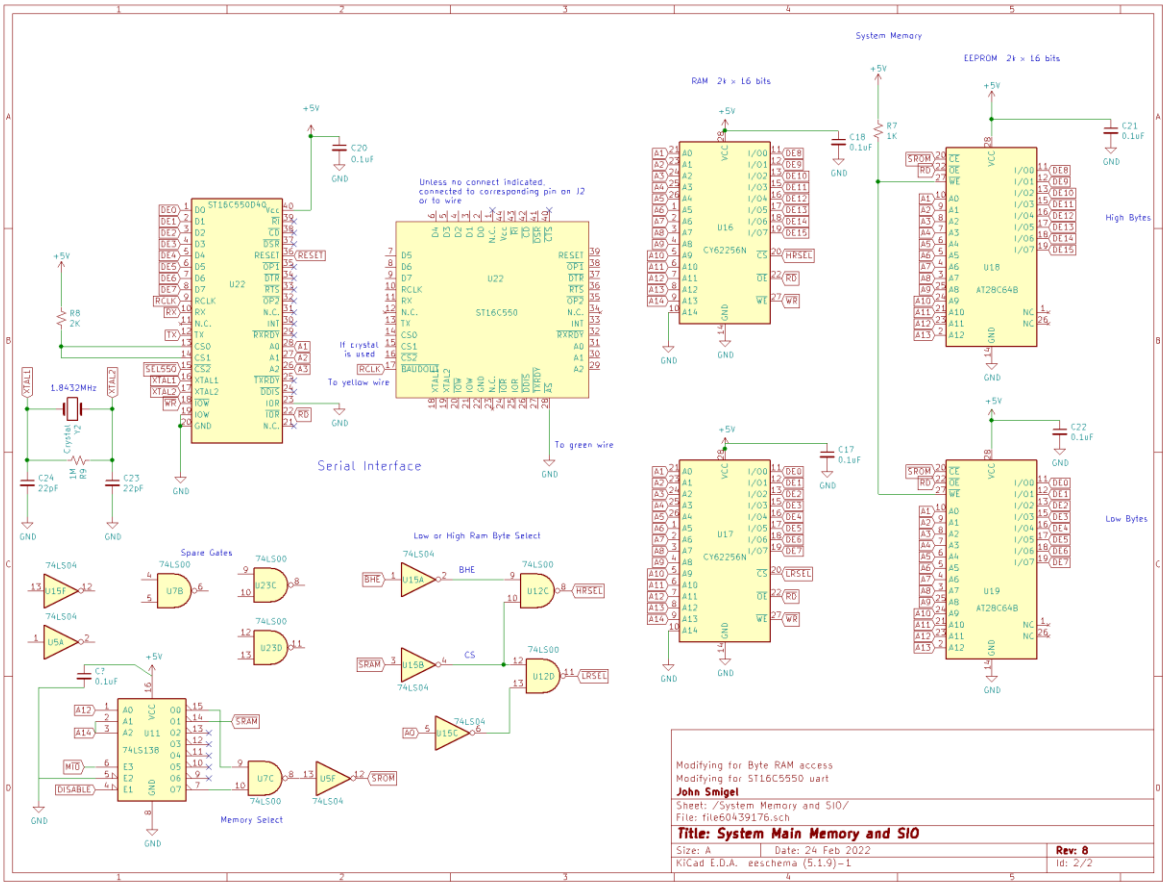


Figure 4-2. *Circuit Design Main System Components*

At the top left is the clock chip. To the right of that is the 8086-4 that outputs to both latches for the 16 address lines and transceiver chips for interfacing the internal data bus with external devices (UART and memory). The transceiver chips were not part of the min system design, but I found I needed them, perhaps due to reduced quality of the 44-year-old 8086-4 signal outputs. The 8259A interrupt controller was removed and not tested because it was not needed. To the lower left are 2 logic circuits to help the processor interface with external chips. I modified these circuits slightly from the kit design. The top is a wait state generator that inserts a couple of wait states when the UART and the EEPROM devices are accessed. This outputs a ready signal to the clock chip that synchronizes it with the clock and passes it to the processor. I trouble with wiring problems causing the processor to hang due to ready (RDY) getting stuck low. I don't know if the waiting is required for the faster devices I ended up using.

Below the wait-state/RDY logic is logic that generates a DISABLE signal that only allows RAM and EEPROMs to be enabled when they absolutely need to be. This is to prevent bus contention. As in the case of the wait-state generator, I don't currently know if the DISABLE

logic is required for the more modern RAM and EEPROM. I modified the DISABLE logic from the kit design to ensure that the RAM is enabled during writes to it. The original logic only insured that the devices were enabled during reads. Only one NAND gate (U12B) and one inverter (U5E) were added. The LS138 chips are demultiplexers to convert the memory and IO addresses to select the desired devices.


System Memory
EEPROM 2k x 16 bits
RAM 2k x 16 bits
High Bytes
Low Bytes
Serial Interface
Unless no connect indicated, connected to corresponding pin on J2 or to wire
If crystal is used
To yellow wire
To green wire
1.8432MHz
Spare Gates
Low or High Ram Byte Select
Memory Select
Modifying for Byte RAM access
Modifying for ST16C5550 uart
John Smigel
Sheet: /System Memory and SIO/
File: file60439176.sch
Title: System Main Memory and SIO
Size: A
Date: 24 Feb 2022
Rev: 8
KiCad E.D.A. eeschema (5.1.9)-1
Id: 2/2

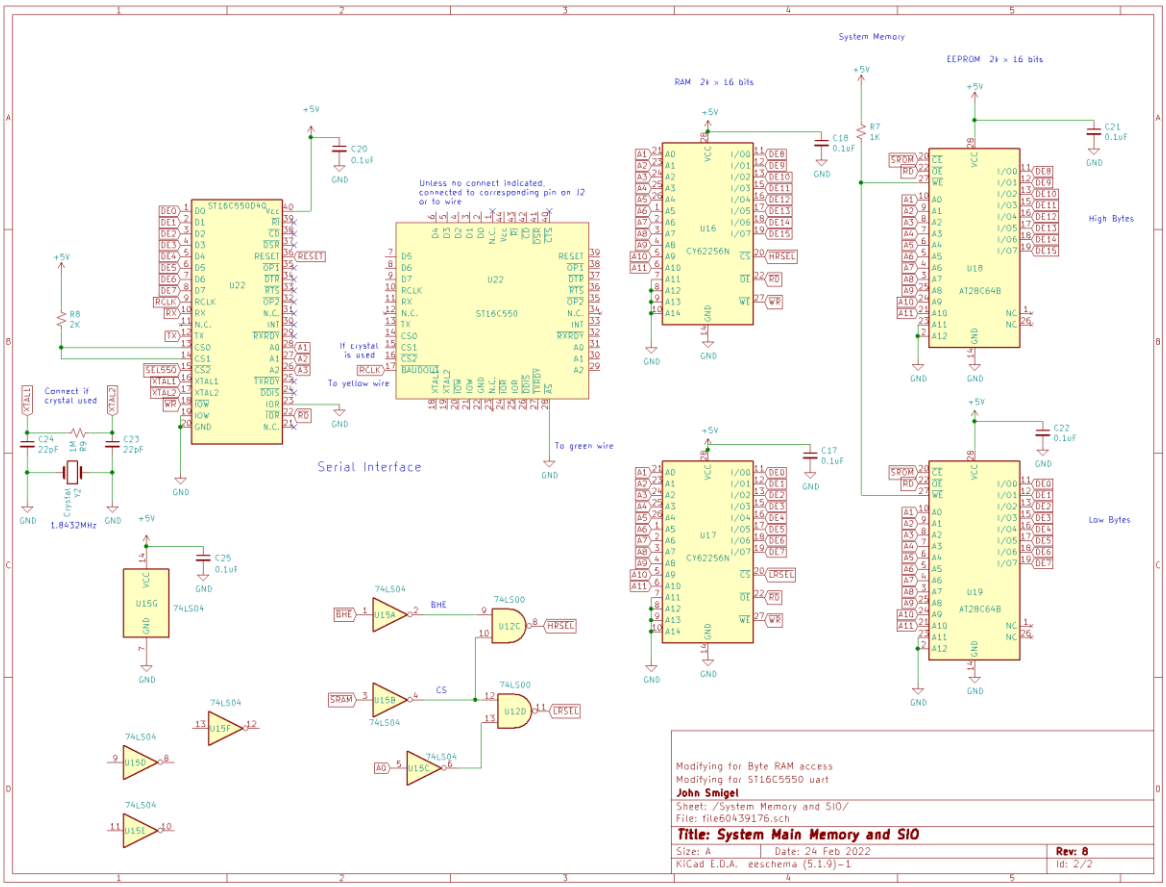


Figure 4-3. *Uart (serial interface) and System Memory (RAM and EEPROM)*

Figure 4-3 shows the serial interface and memory used in the system. I replaced the old kit USART with a slightly more modern ST15550C UART chip in PLCC44 format. The two blocks in the upper left show the 40-pin adapter and the chip itself. An oscillator and counter chip (below the UART and adapter in diagram) were originally used to generate an external clock input to the UART. I switched to using an external crystal with the UART's internal oscillator circuitry because it is slightly simpler and worked ok. Only the external oscillator or clock crystal is needed, not both. If the internal oscillator is used, the 16xbaud rate clock output of the chip must be routed to the receiver clock input, RCLK.

There are 3 left-over NAND gates and 2 inverter gates that are not being used.

The RAM and EEPROM memory chips and connections are shown on the right. Only the lower 11 address input lines of the chips (2K bytes) are currently used.

Figure 4-4. *Original Kit Components*

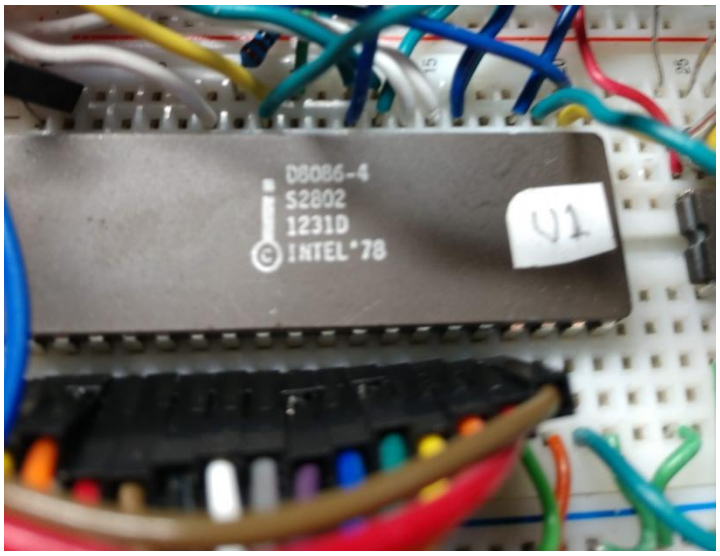


Figure 4-5. *D8086-4 Processor, U1, Installed in Prototyping Breadboard*

Appendix A. Parts List

In case anyone else is crazy enough to want to try this themselves. Here is all you need, and more. Notes: get the appropriate crystal speed for the 8086 version used. The specific multimeter and soldering stations listed are good ones, and I recommend them, but they are not critical and less expensive ones can be used, if needed. The reset switch and power supply listed are not the exact ones I used (I had salvaged them from some other old device).

Unit Labels	Part Number	Description	Cost Per 1	Qty	Total Cost	Link
U1	8086	8086 Microprocessor	$3.99	1	$3.99	Ebay: https://www.ebay.com/itm/162252402922
U2	8284	Clock Generator	$9.95	1	$9.95	https://www.jameco.com/z/8284A-Major-E
U3, U14	74LS74N	Dual D-type Flip Flop	$0.75	2	$1.50	https://www.mouser.com/ProductDetail/T
U5, U15	74ALS04BN	Hex Inverters	$0.89	2	$1.78	https://www.mouser.com/c/?q=74ALS04BI
U6	8259A	Interrupt Controller(optional)	$7.96	1	$7.96	https://www.ebay.com/itm/371506011539 0&mkcid=2&itemid=371506011539&target
U7,U12	74ALS00	Quad 2-input NAND Gates	$1.09	2	$2.18	https://www.mouser.com/c/?q=SN74ALS0
U8,9	74LS373	8-bit Latch for address bus	$1.04	2	$2.08	https://www.mouser.com/c/?q=SN74LS373
U10-11	74AS138N	3-line to 8-line decoders/demultiplexers	$4.64	2	$9.28	https://www.mouser.com/c/?q=SN74AS13
U13	8251A	UART Serial Interface	$3.96	1	$3.96	https://www.mouser.com/ProductDetail/M
		PLCC to DIP Adaptor	$8.00	1	$8.00	https://www.mouser.com/ProductDetail/M

U16-17	CY62256N	Static Ram 32K x 8	$8.50	2	$17.00	https://www.ebay.com/itm/253714866188 312%26itm%3D253714866188%26pmt%3D
U18-19	AT28C64B	8K x 8 EEPROM	$4.76	2	$9.52	https://www.microchip.com/en-us/produc
U20-21	74LS245	8-bit bidirectional Driver	$0.81	2	$1.62	https://www.mouser.com/ProductDetail/T
R1,2,4,5, 6,7		1K Resistor	$0.36	6	$2.16	https://www.mouser.com/c/?q=1k%20resi
D1	1N914	Diode	$0.10	1	$0.10	https://www.mouser.com/c/semiconducto
SW1		Reset Switch	$6.34	1	$6.34	https://www.mouser.com/ProductDetail/E
Y2		1.8432 MHz Crystal for uart	$0.85	1	$0.85	https://www.mouser.com/ProductDetail/A
Y1		12 MHz Crystal for 8086-4	$0.54	1	$0.54	https://www.mouser.com/c/passive-compo
R3		560K Resistor	$0.13	1	$0.13	https://www.mouser.com/c/?q=560K%20r
C3		1uF Capacitor	$0.87	1	$0.87	https://www.mouser.com/ProductDetail/V
C7		22uF Capacitor	$1.03	1	$1.03	https://www.mouser.com/ProductDetail/C
C1,2,4,5, 6,8-22		0.1uF Capacitor	$0.36	20	$7.20	https://www.mouser.com/ProductDetail/K
R9		1M Resistor	$0.12	1	$0.12	https://www.mouser.com/ProductDetail/Y
C23-24		22pF Capacitor	$0.34	2	$0.68	https://www.mouser.com/ProductDetail/V
		Development Tools				
		STK600 Microcontroller Development Kit	$293.99	1	$293.99	https://www.microchip.com/en-us/develop

		Ribbon Wires	$6.98	5	$34.90	https://www.amazon.com/dp/B01EV70C78
		Spool Wires	$15.99	1	$15.99	https://www.amazon.com/TUOFENG-Wire-spools/dp/B07TX6BX47/ref=asc_df_B07TX6
		Oscilloscope	$419.00	1	$419.00	https://www.tequipment.net/Rigol/DS1054
		Soldering Iron station	$104.95	1	$104.95	https://www.amazon.com/Hakko-FX888D-20&linkCode=df0&hvadid=198093934741&
		Logic Analyzer Probe and Software	$159.99	1	$159.99	https://www.amazon.com/dp/B07D35FNYL
		Wire Stripper and Cutter	$10.95	1	$10.95	https://www.amazon.com/Jonard-JIC-2030
		PLCC Chip Puller	$7.49	1	$7.49	https://www.amazon.com/a06111000ux00 20&linkCode=df0&hvadid=241986316970&
		RS-232 to USB adapter cable	$9.99	1	$9.99	https://www.amazon.com/dp/B0758B6MK
		UART to USB adapter cable	$14.88	1	$14.88	https://www.amazon.com/dp/B08BNDLQS
		Digital Multimeter	$399.00	1	$399.00	https://www.amazon.com/Fluke-FLUKE-87-20&linkCode=df0&hvadid=309793588525&
		Breadboard Components				
		11x17 Cookie Sheet	$8.12	1	$8.12	https://www.amazon.com/Good-Cook-040

		5V, 20W Power Supply	$12.99	1	$12.99	https://www.amazon.com/dp/B00MRGKPH
		On/Off Switch	$5.99	1	$5.99	https://www.amazon.com/HiLetgo-AC125V 855f7950e7dd&pd_rd_wg=5
		Power LED	$6.99	1	$6.99	https://www.amazon.com/eBoot-Pieces-Er
		Breadboard Strips	$6.99	10	$69.90	https://www.mouser.com/ProductDetail/D
		Breadboard Power Rails	$1.75	16	$28.00	https://www.mouser.com/ProductDetail/B
		Quick Release Socket, 28 pin	$11.19	1	$11.19	https://www.amazon.com/dp/B094WZQG3 21df97a15e1d&s=electronics&spLa=ZW5jc
		Software Tools				
		NASM Assember (I used rev. 2.15.05)		1	$0.00	https://www.nasm.us/
		Microchip Studio		1	$0.00	https://www.microchip.com/en-us/tools-re
		PuTTY Terminal Emulator		1	$0.00	https://www.putty.org/
		Octave Open Version of MATLAB		1	$0.00	https://www.gnu.org/software/octave/dow
		KiCad Circuit Design Tool		1	$0.00	https://www.kicad.org/download/
					$1,713.15	

Appendix B. Monitor Code Listing

Here is a listing of the assembly language test code I used. Comments show higher level language commands that were converted to assembly language by a compiler I don't have. Assembled using the NASM assembler.

```
   1                                  ; Full version of monitor
   2                                  ; 8086TestUart5.asm  2/23/2022
   3                                  ; Modified to use a ST16C2552 or similar uart
   4                                  cpu     8086
   5                                  ; DEMO86
   6                                  ;       1. Utility Section              Global Flags, equates
   7                                  ;       2. Initialized Data Section     Command Table and constants
   8                                  ;       4. Code         Main program and functions for initial testing
   9                                  ;       5. Boot Strap and RAM variables     Jump to main from reset
  10
  11                                  ; Utilities Section
  12
  13                                  TRUE equ     0FFh
  14                                  FALSE        equ     000h
  15
  16                                  ; Declarations Section
  17
  18                                  ASCR                equ     0Dh     ;Carriage return
  19                                  ASLF                equ     0Ah     ;Line Feed
  20                                  ASBL                equ     020h    ;Blank or space
  21
  22                                  ; Initialized data in EEPROM
  23                                  ORG 0
  24                                  section .text
  25                                  ; sections: .text is for code, .data is for initialized data, .bss is for
uninitialized variables
  26                                  ;section .data ; note: putting in a data section moves this data to the
end of the .text section in the binary binary file
  27                                  ; below initializations give a warning: attempt to initialize in a nobits
section, but I don't want it at the end, for now
  28
  29                                  ; Note: bytes in output listing are ordered low high so address BA00h
looks like 00BAh
  30 00000000 303132333435363738-     ASCII                                   db
        '0123456789ABCDEF'
  30 00000009 39414243444546
  31 00000010 475344              SIOCMND                                 db      'GSD'
```

```
  32 00000013 44454D4F2D38362056-     @@LONG$CONSTANT$00A9H  db       'DEMO-86 VER
1.2'
  32 0000001C 455220312E32
  33 00000022 535542535449545554-     @@LONG$CONSTANT$0013H  db       'SUBSTITUTE
FAILED'
  33 0000002B 45204641494C4544
  34
  35                          ; want stack start absolute address to be 17e0h; 101hx10+7d0h = 17e0h
  36 00000033 D007                @@STACK$OFFSET                  dw 07D0h
  37 00000035 0101                @@STACK$FRAME                   dw 00101h
  38 00000037 0101                @@DATA$FRAME                    dw 00101h
  39
  40 00000039 00<rep 83h>             TIMES 0BCh-($-$$) DB 0 ; start code at location BCh
(relative to physical code segment start of FF60h
  41
  42                          ;Main program where the program starts.  Initializes
  43                          ;IO devices, waits for a key depression, and prints out the sign on
message.
  44                          ;It then jumps to 'NEXT$COMMAND' and waits for input of a command
  45                          ;initialize IO devices
  46 000000BC FA                  CLI ; clear interrupts
  47                          ; Segment registers SS,CS, and DS are Stack, code, and data segment 16
most significant bits of page address
  48                          ; the full 20 bit address is obtained by shifting the segment left 4 bits
and adding the offset
  49                          ; stack grows downward, new data at lower memory locations,  push
decrements stack pointer and points to last value pushed
  50 000000BD B80101              MOV AX, word 0101h
  51 000000C0 8ED0                MOV         SS,AX
  52                          ;MOVSP,[CS:@@STACK$OFFSET]
  53                          ;MOVBP,SP
  54 000000C2 2E8E1E[3700]        MOV    DS,[CS:@@DATA$FRAME]
  55                          ;STI
  56
  57                          MAINPROGRAM:
  58 000000C7 B8D007              MOV        AX,word 7D0h ;STACKPTR = 07D0h
  59 000000CA 89C4                MOV         SP,AX
  60 000000CC 89E5                MOV BP,SP
  61                          ;CALL INTINIT
  62 000000CE E88D00              CALL        SIOINIT
  63                          ;call SIO$OUT$STRING(@('DEMO-86 VER 1.2'),15);
  64 000000D1 2E8D06[1300]                        LEA
       AX,CS:@@LONG$CONSTANT$00A9H;
  65 000000D6 0E                  PUSH CS
```

```
66 000000D7 50                  PUSH AX
67 000000D8 B00F                MOV                     AL,byte 0Fh
68 000000DA 50                  PUSH AX
69 000000DB E80301              CALL        SIOOUTSTRING
70 000000DE E83803              CALL        SIONEWLINE
71 000000E1 E82B01              CALL        SIOGETCHAR
72
73                      NEXTCOMMAND:
74 000000E4 B8D007              MOV         AX,word 7D0h ;STACKPTR = 07D0h
75 000000E7 89C4                MOV         SP,AX
76 000000E9 2E8E1E[3700]            MOV     DS,[CS:@@DATA$FRAME]
77                      a@58:
78 000000EE B0FF                MOVAL,byte 0FFh
79 000000F0 D0D8                RCR AL,1
80 000000F2 7328                JNB   ERROR
81                      ;do while TRUE;
82 000000F4 B8D007              MOV         AX,word 7D0h ;STACKPTR = 07D0H;
83 000000F7 89C4                MOV         SP,AX
84                      ;do case GET$CMND;
85 000000F9 E8BD04              CALL        GETCMND
86 000000FC B400                MOV         AH,byte 00h
87 000000FE 89C3                MOV         BX,AX
88 00000100 D1E3                SHL  BX,1
89 00000102 2EFFA7[1601]            JMP     [CS:a@60+BX];
90                      a@25:
91 00000107 E81D03              CALL        SIOGO
92 0000010A EBE2                JMP a@58
93                      a@26:
94 0000010C E83F03              CALL        SIOEXAMMEM
95 0000010F EBDD                JMP a@58
96                      a@27:
97 00000111 E8EB03              CALL        SIODISPLAY
98 00000114 EBD8                JMP a@58
99                      a@60:
100 00000116 [0701]             DWa@25;
101 00000118 [0C01]             DWa@26
102 0000011A [1101]             DWa@27
103
104
105                      ERROR:
106                      ;Handles all errors and outputs '*' to the 8251.
107 0000011C 2E8E1E[3700]           MOV   DS,[CS:@@DATA$FRAME]
108 00000121 B8D007             MOV         AX,word 7D0h ;STACKPTR = 07D0H
109 00000124 89C4               MOV         SP,AX
```

```
110                     ;call SIO$OUT$CHAR('*');
111 00000126 B02A           MOV     AL,byte 2Ah
112 00000128 50             PUSH    AX
113 00000129 E86100         CALL    SIOOUTCHAR
114 0000012C EBB6           JMP     NEXTCOMMAND
115
116
117                     INTINIT:
118                     ; Initialize the 8259A interrupt controller
119 0000012E 55             PUSH BP
120 0000012F 89E5           MOV     BP,SP
121                     ; Procedure; output(INT$STAT$PORT)=INT$ICW1;
122 00000131 B037           MOV     AL,37h
123 00000133 E600           OUT     0h,AL
124 00000135 B001           MOV     AL,01h
125 00000137 50             PUSH AX
126 00000138 E81000         CALL DELAY
127                     ; Output(INT$MASK$PORT)=INT$ICW2;
128 0000013B B000           MOV     AL,00h
129 0000013D E602           OUT     2h,AL
130 0000013F B001           MOV     AL,01h
131 00000141 50             PUSH AX
132 00000142 E80600         CALL DELAY
133                     ; output(INT$MASK$PORT)=INT$ICW4;
134 00000145 B001           MOV     AL,01h
135 00000147 E602           OUT     2h,AL
136 00000149 5D             POP  BP
137 0000014A C3             RET
138                     ;INTINIT    ENDP
139
140
141                     ; SERIAL MODULE - Implement Serial Mode Command Functions
142
143                     DELAY:
144                     ; Delays by input number of loop cycles
145 0000014B 55             PUSHBP
146 0000014C 89E5           MOV     BP,SP
147                     ; procedure(C);
148                     ;do while C > 0;
149                     ;C equ 0
150                     a@28:
151 0000014E 807E0400       CMP     [BP+4],byte 00h
152 00000152 7606           JBE  a@29
153                     ; C = C - 1;
```

```
  154 00000154 806E0401            SUB       [BP+4],byte 01h
  155                       ;end do
  156 00000158 EBF4                JMP a@28
  157                       a@29:
  158 0000015A 5D                 POP  BP
  159 0000015B C20200              RET       2h
  160                       ;DELAY       ENDP
  161
  162
  163                       SIOINIT:
  164                       ; initializes the USART to command: RTS active, recv & xmit enabled,
  165                       ; mode: 1 stop bit, parity disabled, 8 data bits, X64 baud rate
  166                       ; A0 determines if output goes to high or low byte (1=high), A1 to A3
addrress uart
  167                       ; internal register
  168 0000015E 55                 PUSH BP
  169 0000015F 89E5                MOV       BP,SP
  170 00000161 B400                MOV AH,00h;  probably don't need to clear upper byte
  171 00000163 B080                MOV       AL,80h ; Baud rate divisor bit in LCR
  172 00000165 E626                OUT       26h,AL
  173 00000167 B002                MOV       AL,2h
  174 00000169 50                 PUSH AX
  175 0000016A E8DEFF              CALL      DELAY
  176 0000016D B00C                MOV       AL,0ch; was 4 for external oscillator, now 0ch for
crystal at 1.8432MHz
  177 0000016F E620                OUT20h,AL; DLL
  178 00000171 B002                MOV       AL,2h ; delay 2 loops
  179 00000173 50                 PUSH AX
  180 00000174 E8D4FF              CALL      DELAY
  181 00000177 B000                MOV       AL,00h
  182 00000179 E622                OUT       22h,AL; DLM
  183 0000017B B002                MOV       AL,2h
  184 0000017D 50                 PUSH      AX
  185 0000017E E8CAFF              CALL      DELAY
  186 00000181 B003                MOV       AL,03h
  187 00000183 E626                OUT       26h,AL
  188 00000185 B002                MOV       AL,2h
  189 00000187 50                 PUSH AX
  190 00000188 E8C0FF              CALL      DELAY
  191 0000018B 5D                 POP  BP
  192 0000018C C3                 RET
  193                       ;SIOINIT       ENDP
  194
  195
```

```
  196                   SIOOUTCHAR:
  197                   ;outputs the input parameter to the USART output
  198                   ;port when USART is ready for output (xmit buffer empty)
  199 0000018D 55             PUSH BP
  200 0000018E 89E5           MOV     BP,SP
  201                   ;procedure(C); declare C BYTE; do while (input(SIO$STAT$PORT) and
SIO$TXRDY) = 0;
  202                   ;C    equ 0
  203                   a@30:
  204 00000190 E42A           IN   AL,2ah; LSR
  205 00000192 A820           TEST    AL,20h
  206 00000194 74FA           JZ   a@30
  207                   ;output(SIO$DATA$PORT) = C;
  208 00000196 8A4604         MOV     AL,[BP+4]
  209 00000199 E620           OUT     20h,AL; THR
  210 0000019B 5D             POP  BP
  211 0000019C C20200         RET     2h
  212                   ;SIOOUTCHAR     ENDP
  213
  214
  215                   SIOOUTBYTE:
  216                   ;Outputs a byte to the USART in the 2 digit ascii equivalence
  217                   ;of the byte input parameter
  218 0000019F 55             PUSH BP
  219 000001A0 89E5           MOV     BP,SP
  220                   ;procedure(B);
  221                   ;DECLARE B BYTE
  222                   ;B    equ 0
  223                   ;call SIO$OUT$CHAR(ASCII(SHR(B,4) AND 0FH));
  224 000001A2 8A5E04         MOV     BL,[BP+4] ; B address is stored in stack pointer,SP,
SP-2 (BP pushed) was moved to BP
  225 000001A5 B104           MOV     CL,04h
  226 000001A7 D2EB           SHR     BL,CL
  227 000001A9 80E30F         AND     BL,0Fh
  228 000001AC B700           MOV     BH,00h
  229 000001AE 2EFFB7[0000]       PUSH word [CS:ASCII+BX]; minimum size you can push is
word
  230 000001B3 E8D7FF         CALL    SIOOUTCHAR
  231                   ;call SIO$OUT$CHAR(ASCII(B AND 0FH));
  232 000001B6 8A5E04         MOV     BL,[BP+4]
  233 000001B9 80E30F         AND     BL,0Fh
  234 000001BC B700           MOV             BH,00h
  235 000001BE 2EFFB7[0000]       PUSH word [CS:ASCII+BX]
  236 000001C3 E8C7FF         CALL    SIOOUTCHAR
```

```
 237 000001C6 5D                POP         BP
 238 000001C7 C20200              RET       2h
 239                    ;SIOOUTBYTE         ENDP
 240
 241
 242                    SIOOUTWORD:
 243                    ;outputs a word as 4 ascii digits to the usart output port
 244 000001CA 55              PUSH          BP
 245 000001CB 89E5              MOV         BP,SP
 246                    ;procedure(w);
 247                    ;declare w word;
 248                    ;call SIO$OUT$BYTE(HIGH(W));
 249                    ;W    equ     0
 250 000001CD 8B4604              MOV       AX,[BP+4]
 251 000001D0 88E0              MOV         AL,AH
 252 000001D2 50                PUSH        AX
 253 000001D3 E8C9FF              CALL      SIOOUTBYTE
 254                    ;call SIO$OUT$BYTE(LOW(W));
 255 000001D6 8B4604              MOV       AX,[BP+4]
 256 000001D9 50                PUSH        AX
 257 000001DA E8C2FF              CALL      SIOOUTBYTE
 258 000001DD 5D                POP         BP
 259 000001DE C20200              RET               2h
 260                    ;SIOOUTWORD       ENDP
 261
 262
 263                    SIOOUTSTRING:
 264                    ;outputs a string pointed to by the first param of length
 265                    ;second param
 266 000001E1 55              PUSH BP
 267 000001E2 89E5              MOV         BP,SP
 268                    ;procedure (STR$PTR,N);
 269                    ;declare STR$PTR POINTER, (N,I) BYTE,STR BASED STR$PTR (1) BYTE,
 270                    ;do i=0 to n-1;
 271                    ;N    equ     2
 272 000001E4 C606[8308]00          MOV   [I],byte 00h
 273                    a@32:
 274                    ; push decrements stack before writing to it
 275 000001E9 8A4604              MOV       AL,[BP+4] ; assuming # bytes param is 2 bytes and
[BP+1} points to LSByte
 276 000001EC 2C01              SUB AL,1h
 277 000001EE 3806[8308]            CMP               [I],AL
 278 000001F2 7717              JA    a@33
 279                    ;call SIO$OUT$CHAR(STR(I));
```

```
 280 000001F4 A0[8308]            MOV       AL,[I]
 281 000001F7 B400                MOV       AH,00h
 282 000001F9 89C6                MOV       SI,AX; SI is string index
 283                      ; below loads the page and offset address on the stack
 284 000001FB C45E06              LES       BX,[BP+6]; point to 32 bit address pair pushed on
the stack first (page, offset)
 285 000001FE 26FF30              PUSH word [ES:BX+SI]; minimum size can push is word
 286 00000201 E889FF              CALL      SIOOUTCHAR
 287                      ;end do
 288 00000204 8006[8308]01        ADD    [I],byte 01h
 289 00000209 75DE                JNZ a@32
 290                      a@33:
 291 0000020B 5D                 POP  BP
 292 0000020C C20600              RET       6h
 293                      ;SIOOUTSTRING     ENDP
 294
 295
 296                      SIOGETCHAR:
 297                      ;inputs 1 char from the USART input port and returns in the
 298                      ;global variable 'CHAR'
 299 0000020F 55                 PUSH BP
 300 00000210 89E5                MOV       BP,SP
 301                      ;do while (INPUT(SIO$STAT$PORT) AND SIO$RXRDY)=0;
 302                      a@34:
 303 00000212 E42A                IN    AL,2ah; LSR
 304 00000214 A801                TEST      AL,01h
 305 00000216 74FA                JZ a@34
 306                      ;char = input(SIO$DATA$PORT) AND 07FH;
 307 00000218 E420                IN    AL,20h; RHR
 308 0000021A 247F                AND       AL,7Fh
 309 0000021C A2[8008]            MOV       [CHAR],AL
 310                      ;if CHAR >=ASBL then
 311 0000021F 3C20                CMP       AL,20h
 312 00000221 7207                JB a@1
 313                      ;call SIO$OUT$CHAR(CHAR);
 314 00000223 FF36[8008]          PUSH word [CHAR]
 315 00000227 E863FF              CALL      SIOOUTCHAR
 316                      a@1:
 317 0000022A 5D                 POP  BP
 318 0000022B C3                 RET
 319                      ;SIOGETCHAR       ENDP
 320
 321
 322                      SIOVALIDHEX:
```

```
323                     ;tests if the input param is a valid ascii
324                     ;hex digit and returns true as the value of the procedure
325                     ;if so and false if not
326 0000022C 55               PUSH BP
327 0000022D 89E5             MOV     BP,SP
328                     ;procedure (H) BYTE;
329                     ;DECLARE H BYTE;
330                     ;if ('0'<=H AND H<='9') OR ('A'<=H AND H<='F') then
331 0000022F B130             MOV     CL,30h
332 00000231 8A4604           MOV     AL,[BP+4]
333 00000234 38C1             CMP     CL,AL
334 00000236 B0FF             MOV     AL,0FFh ; hex value needs to start with number
335 00000238 7601             JBE  $+3h
336 0000023A 40             INC   AX
337 0000023B 8A4E04           MOV     CL,[BP+4]
338 0000023E 50             PUSH AX
339 0000023F 80F939           CMP             CL,39h
340 00000242 B0FF             MOV     AL,0FFh
341 00000244 7601             JBE  $+3h
342 00000246 40             INC   AX
343 00000247 5A             POP  DX
344 00000248 20D0             AND     AL,DL
345 0000024A B241             MOV     DL,41h
346 0000024C 50             PUSH AX
347 0000024D 38CA             CMP     DL,CL
348 0000024F B0FF             MOV     AL,0FFh
349 00000251 7601             JBE  $+3h
350 00000253 40             INC   AX
351 00000254 50             PUSH AX
352 00000255 80F946           CMP     CL,46h
353 00000258 B0FF             MOV     AL,0FFh
354 0000025A 7601             JBE  $+3h
355 0000025C 40             INC   AX
356 0000025D 59             POP  CX
357 0000025E 20C8             AND     AL,CL
358 00000260 59             POP  CX
359 00000261 08C8             OR  AL,CL
360 00000263 D0D8             RCR     AL,1
361 00000265 7304             JNB a@2
362                     ;return true;
363 00000267 B0FF             MOV     AL,0FFh
364 00000269 EB02             JMP a@31
365                     a@2:
366                     ;else
```

```
367                     ; return false
368 0000026B B000           MOV     AL,00h
369                     a@31:
370 0000026D 5D           POP  BP
371 0000026E C20200         RET     2h
372                     ;SIOVALIDHEX    ENDP
373
374
375                     SIOHEX:
376                     ;converts the input param from ascii to its binary
377                     ;equivalent and returns it as the value of the
378                     ;procedure. No check for input validity.
379 00000271 55           PUSH BP
380 00000272 89E5           MOV     BP,SP
381                     ;procedure (C) BYTE
382                     ;declare C BYTE;
383                     ;C = C - 30H;
384 00000274 8A4604         MOV     AL,[BP+4]
385 00000277 2C30           SUB AL,30H
386 00000279 884604         MOV     [BP+4],AL
387                     ;if C <=9 then return C;
388 0000027C 3C09           CMP     AL,09h
389 0000027E 7705           JA  a@4
390 00000280 8A4604         MOV     AL,[BP+4]
391 00000283 EB05           JMP a@3
392                     a@4:
393                     ;else return C-7;
394 00000285 8A4604         MOV     AL,[BP+4]
395 00000288 2C07           SUB AL,07h
396                     a@3:
397 0000028A 5D           POP  BP
398 0000028B C20200         RET     2h
399                     ;SIOHEX     ENDP
400
401
402                     SIOGETBYTE:
403                     ;scans the input stream for a valid byte
404 0000028E 55           PUSH BP
405 0000028F 89E5           MOV     BP,SP
406                     ;procedure BYTE;
407                     ;declare B BYTE;
408                     ;if not(SIO$VALID$HEX(CHAR)) then go to ERROR;
409 00000291 FF36[8008]       PUSH word [CHAR]
410 00000295 E894FF         CALL    SIOVALIDHEX
```

```
411 00000298 F6D0              NOT         AL
412 0000029A D0D8              RCR         AL,1
413 0000029C 7303              JNB $+5h
414 0000029E E97BFE            JMP         ERROR
415                     ;B = 0;
416 000002A1 C606[8208]00      MOV   [B],byte 00h
417                     ;do while SIO$VALID$HEX(CHAR);
418                     a@36:
419 000002A6 FF36[8008]        PUSH word [CHAR]
420 000002AA E87FFF            CALL        SIOVALIDHEX
421 000002AD D0D8              RCR         AL,1
422 000002AF 731A              JNB a@37
423                     ;B = SHL(B,4) + SIO$HEX(CHAR);
424 000002B1 A0[8208]          MOV       AL,[B]
425 000002B4 B104              MOV         CL,04h
426 000002B6 D2E0              SHL AL,CL
427 000002B8 50                PUSHAX
428 000002B9 FF36[8008]        PUSH word [CHAR]
429 000002BD E8B1FF            CALL        SIOHEX
430 000002C0 59                POP   CX
431 000002C1 00C8              ADD         AL,CL
432 000002C3 A2[8208]          MOV       [B],AL
433 000002C6 E846FF            CALL        SIOGETCHAR
434 000002C9 EBDB              JMP         a@36
435                     a@37:
436                     ;if CHAR = ASCR OR CHAR = ASBL OR CHAR = ',' then return B;
437 000002CB A0[8008]          MOV       AL,[CHAR]
438 000002CE 3C0D              CMP         AL,0Dh
439 000002D0 B0FF              MOV         AL,0FFh
440 000002D2 7401              JZ    $+3h
441 000002D4 40                INC   AX
442 000002D5 50                PUSH        AX
443 000002D6 803E[8008]20      CMP   [CHAR],byte 20h
444 000002DB B0FF              MOV         AL,0FFh
445 000002DD 7401              JZ    $+3h
446 000002DF 40                INC   AX
447 000002E0 59                POP   CX
448 000002E1 08C8              OR  AL,CL
449 000002E3 50                PUSHAX
450 000002E4 803E[8008]2C      CMP   [CHAR],byte 2Ch
451 000002E9 B0FF              MOV         AL,0FFh
452 000002EB 7401              JZ    $+3h
453 000002ED 40                INC   AX
454 000002EE 59                POP   CX
```

```
455 000002EF 08C8               OR   AL,CL
456 000002F1 D0D8               RCR          AL,1
457 000002F3 7203               JB    $+5h
458 000002F5 E924FE             JMP          ERROR
459 000002F8 A0[8208]           MOV          AL,[B]
460 000002FB 5D                 POP  BP
461 000002FC C3                 RET
462                     ;SIOGETBYTE          ENDP
463
464
465                     SIOGETWORD:
466                     ;reads chars from the input port and evaluates an expression
467                     ;consisting of '+-' and operands of hex numbers
468 000002FD 55                 PUSHBP
469 000002FE 89E5               MOV          BP,SP
470                     ;procedure word;
471                     ;declare (SAVE,W) WORD, OPER BYTE;
472                     ;OPER = '+';
473 00000300 C606[8708]2B       MOV   [OPER],byte 2Bh
474                     ;W = 0;
475 00000305 C706[8C08]0000     MOV [W],word 0000h
476                     ;SAVE = 0;
477 0000030B C706[8808]0000     MOV [SAVE],word 0000h
478                     ;do while TRUE;
479                     a@38:
480 00000311 B0FF               MOV          AL,0FFh
481 00000313 D0D8               RCR          AL,1
482 00000315 7203               JB    $+5h
483 00000317 E9A900             JMP          a@39
484                     ;do while SIO$VALID$HEX(CHAR);
485                     a@40:
486 0000031A FF36[8008]         PUSH word [CHAR]
487 0000031E E80BFF             CALL         SIOVALIDHEX
488 00000321 D0D8               RCR          AL,1
489 00000323 731C               JNB a@41
490                     ;SAVE = SHL(SAVE,4) + DOUBLE(SIO$HEX(CHAR));
491 00000325 A1[8808]           MOV          AX,[SAVE]
492 00000328 B104               MOV          CL,04h
493 0000032A D3E0               SHLAX,CL
494 0000032C 50                 PUSHAX
495 0000032D FF36[8008]         PUSH word [CHAR]
496 00000331 E83DFF             CALL         SIOHEX
497 00000334 B400               MOV          AH,00h
498 00000336 59                 POP   CX
```

```
499 00000337 01C8              ADD         AX,CX
500 00000339 A3[8808]            MOV       [SAVE],AX
501 0000033C E8D0FE              CALL      SIOGETCHAR
502 0000033F EBD9              JMP a@40
503                       a@41:
504                       ;if OPER='+' then
505 00000341 803E[8708]2B         CMP      [OPER],byte 2Bh
506 00000346 7509              JNZ a@8
507                       ;W = W + SAVE;
508 00000348 A1[8808]            MOV       AX,[SAVE]
509 0000034B 0106[8C08]           ADD      [W],AX
510 0000034F EB07              JMP a@9
511                       a@8:
512                       ;else
513                       ;W = W - SAVE;
514 00000351 A1[8808]            MOV       AX,[SAVE]
515 00000354 2906[8C08]           SUB      [W],AX
516                       a@9:
517                       ;if CHAR=ASCR OR CHAR=':' OR CHAR=',' OR CHAR=ASBL then return W;
518 00000358 A0[8008]            MOV       AL,[CHAR]
519 0000035B 3C0D              CMP         AL,0Dh
520 0000035D B0FF              MOV         AL,0FFh
521 0000035F 7401              JZ    $+3h
522 00000361 40                INC    AX
523 00000362 8A0E[8008]           MOV     CL,[CHAR]
524 00000366 50                PUSH AX
525 00000367 80F93A               CMP      CL,3Ah
526 0000036A B0FF              MOV         AL,0FFh
527 0000036C 7401              JZ    $+3h
528 0000036E 40                INC    AX
529 0000036F 5A                POP   DX
530 00000370 08D0                OR  AL,DL
531 00000372 50                PUSH AX
532 00000373 80F92C               CMP      CL,2Ch
533 00000376 B0FF              MOV         AL,0FFh
534 00000378 7401              JZ    $+3h
535 0000037A 40                INC AX
536 0000037B 5A                POP   DX
537 0000037C 08D0                OR  AL,DL
538 0000037E 50                PUSH AX
539 0000037F 80F920               CMP      CL,20h
540 00000382 B0FF              MOV         AL,0FFh
541 00000384 7401              JZ    $+3h
542 00000386 40                INC    AX
```

```
543 00000387 59              POP   CX
544 00000388 08C8              OR  AL,CL
545 0000038A D0D8              RCR         AL,1
546 0000038C 7305              JNB a@10
547 0000038E A1[8C08]            MOV       AX,[W]
548 00000391 5D              POP  BP
549 00000392 C3              RET
550                     a@10:
551                     ;if CHAR='+" or CHAR='-' then
552 00000393 803E[8008]2B          CMP    [CHAR],byte 2Bh
553 00000398 B0FF              MOV         AL,0FFh
554 0000039A 7401              JZ    $+3h
555 0000039C 40              INC    AX
556 0000039D 50              PUSH        AX
557 0000039E 803E[8008]2D          CMP    [CHAR],byte 2Dh
558 000003A3 B0FF              MOV         AL,0FFh
559 000003A5 7401              JZ    $+3h
560 000003A7 40              INC    AX
561 000003A8 59              POP   CX
562 000003A9 08C8              OR  AL,CL
563 000003AB D0D8              RCR         AL,1
564 000003AD 7203              JB   $+5h
565 000003AF E96AFD              JMP        ERROR
566                     ;do
567                     ;OPER=CHAR;
568 000003B2 A0[8008]            MOV       AL,[CHAR]
569 000003B5 A2[8708]            MOV       [OPER],AL
570                     ;SAVE = 0;
571 000003B8 C606[8808]00          MOV   [SAVE],byte 00h
572 000003BD E84FFE              CALL       SIOGETCHAR
573 000003C0 E94EFF              JMP        a@38
574                     a@39:
575 000003C3 5D              POP  BP
576 000003C4 C3              RET
577                     ;SIOGETWORD       ENDP
578
579
580                     SIOGETADDR:
581                     ;accepts a valid address expression
582                     ;consisting of an optional <seg>: and a displacement
583 000003C5 55              PUSH BP
584 000003C6 89E5              MOV         BP,SP
585                     ;procedure(DEFAULT$BASE);
586                     ;DECLARE DEFAULT$BASE WORD;
```

```
  587                     ;ARG1.SEG = DEFAULT$BASE;
  588 000003C8 8B4604             MOV     AX,[BP+4]; assuming BP has default segment base
pushed on stack
  589                     a@5:
  590 000003CB A3[9008]           MOV     [ARG1+2h],AX;
  591                     ;arg1.off = SIO$GET$WORD;
  592 000003CE E82CFF             CALL SIOGETWORD
  593 000003D1 A3[8E08]           MOV     [ARG1],AX;
  594                     ;do while CHAR= ':';
  595 000003D4 803E[8008]3A       CMP   [CHAR],byte 3Ah
  596 000003D9 7508               JNZ a@43
  597 000003DB E831FE             CALL SIOGETCHAR
  598                     ;arg.seg = arg1.off;
  599 000003DE A1[8E08]           MOV     AX,[ARG1];
  600 000003E1 EBE8               JMP a@5
  601                     ;arg.off = SIO$GET$WORD; can't reach this statement, so ignored? yes,
but jmp is to same stmnt
  602                     a@43:
  603 000003E3 5D               POP   BP
  604 000003E4 C20200           RET     2h
  605                     ;SIOGETADDR     ENDP
  606
  607
  608                     SIOBLANKS:
  609                     ;Outputs the number of blanks specified by the input parameter
  610 000003E7 55             PUSH BP
  611 000003E8 89E5             MOV BP,SP
  612                     ;procedure(N);
  613                     ;declare (I,N) BYTE;
  614                     ;do i=1 to N;
  615 000003EA C606[8308]01       MOV   [I],byte 01h
  616                     a@44:
  617 000003EF A0[8308]           MOV     AL,[I]
  618 000003F2 3A4604             CMP     AL,[BP+4]; .N
  619 000003F5 770D               JA    a@45
  620                     ;call SIO$OUT$CHAR(ASBL);
  621 000003F7 B020               MOV     AL,byte 20h
  622 000003F9 50               PUSH AX
  623 000003FA E890FD             CALL    SIOOUTCHAR
  624 000003FD 8006[8308]01       ADD   [I],byte 01h
  625 00000402 75EB               JNZ a@44
  626                     a@45:
  627 00000404 5D               POP   BP
  628 00000405 C20200             RET     2h
```

```
629                     ;SIOBLANKS  ENDP
630
631
632                     SIOCRLF:
633                     ;Outputs a CR and LF to the output port
634 00000408 55              PUSH BP
635 00000409 89E5             MOV       BP,SP
636                     ;call SIO$OUT$CHAR(ASCR);
637 0000040B B00D             MOV       AL,byte 0Dh
638 0000040D 50              PUSH       AX
639 0000040E E87CFD            CALL      SIOOUTCHAR
640                     ;call SIO$OUT$CHAR(ASLF);
641 00000411 B00A             MOV       AL,byte 0Ah
642 00000413 50              PUSH AX
643 00000414 E876FD            CALL      SIOOUTCHAR
644 00000417 5D              POP  BP
645 00000418 C3              RET
646                     ;SIOCRLF      ENDP
647
648                     SIONEWLINE:
649                     ; outputs a new line and 2 blanks
650 00000419 55              PUSH BP
651 0000041A 89E5             MOV       BP,SP
652 0000041C E8E9FF            CALL      SIOCRLF
653 0000041F B002             MOV       AL,byte 02h
654 00000421 50              PUSH AX
655 00000422 E8C2FF            CALL      SIOBLANKS
656 00000425 5D              POP  BP
657 00000426 C3              RET
658                     ;SIONEWLINE         ENDP
659
660                     ;Commands Section (note placeholders here for initial test now)
661
662                     SIOGO:
663                     ;Implements the GO command. User may specify new CS and PC
664 00000427 55              PUSH BP
665 00000428 89E5             MOV       BP,SP
666                     ;CALL SIO$GET$CHAR;
667                     a@6:
668 0000042A E8E2FD            CALL SIOGETCHAR
669                     ;do while CHAR=ASBL;
670 0000042D 803E[8008]20          CMP    [CHAR],byte 20h
671 00000432 74F6              JZ a@6
672                     ;call SIO$GET$ADDR(0);
```

```
673 00000434 B80000              MOV       AX,word 0000h
674 00000437 50                 PUSH AX
675 00000438 E88AFF              CALL SIOGETADDR ; output is in ARG1 far pointer
676                         ;if CHAR<>ASCR then goto error
677 0000043B 803E[8008]0D         CMP    [CHAR],byte 0Dh
678 00000440 7403               JZ    $+5h
679 00000442 E9D7FC              JMP       ERROR
680 00000445 E8C0FF              CALL SIOCRLF
681 00000448 FF16[8E08]           CALL [MEMORYARG1PTR]
682 0000044C 5D                 POP  BP
683 0000044D C3                 RET
684                         ;SIOGO        ENDP
685
686
687                         SIOEXAMMEM:
688                         ;Implements the examine memory command
689 0000044E 55                 PUSH BP
690 0000044F 89E5                MOV         BP,SP
691                         ;declare T BYTE;
692                         a@7:
693 00000451 E8BBFD              CALL SIOGETCHAR
694 00000454 803E[8008]20         CMP [CHAR], byte 20h
695 00000459 74F6               JZ a@7
696                         ;call SIO$GET$ADDR(0);
697 0000045B B80000              MOV       AX,word 0000h
698 0000045E 50                 PUSH AX
699 0000045F E863FF              CALL SIOGETADDR
700                         ;if CHAR<>ASCR then goto ERROR
701 00000462 803E[8008]0D         CMP    [CHAR],byte 0Dh
702 00000467 7403               JZ    $+5h
703 00000469 E9B0FC              JMP       ERROR
704                         ;do while TRUE;
705                         a@50:
706 0000046C B0FF               MOV       AL,byte 0FFh
707 0000046E D0D8               RCR       AL,1
708 00000470 7203               JB    $+5h
709 00000472 E98800              JMP       a@51
710 00000475 E8A1FF              CALL SIONEWLINE
711                         ;call SIO$OUT$WORD(ARG1.OFF)
712 00000478 FF36[8E08]           PUSH     word [ARG1]
713 0000047C E84BFD              CALL SIOOUTWORD
714                         ;call SIO$OUT$CHAR('-')
715 0000047F B02D               MOV       AL,byte 2Dh
716 00000481 50                 PUSH AX
```

```
717 00000482 E808FD              CALL SIOOUTCHAR
718 00000485 C41E[8E08]           LES       BX,[MEMORYARG1PTR]
719 00000489 26FF37              PUSH word [ES:BX]
720 0000048C E810FD              CALL SIOOUTBYTE
721                      ;call SIO$OUTCHAR('-')
722 0000048F B02D                MOV       AL,byte 2Dh
723 00000491 50                 PUSH AX
724 00000492 E8F8FC              CALL SIOOUTCHAR
725 00000495 E877FD              CALL SIOGETCHAR
726                      ;if CHAR<>',' and CHAR<>ASBL and CHAR <> ASCR then
727 00000498 A0[8008]             MOV       AL,[CHAR]
728 0000049B 3C2C                CMP       AL,byte 2Ch
729 0000049D B0FF                MOV       AL,byte 0FFh
730 0000049F 7501                JNZ  $+3h
731 000004A1 40                 INC    AX
732 000004A2 50                 PUSH AX
733 000004A3 803E[8008]20          CMP    [CHAR],byte 20h
734 000004A8 B0FF                MOV       AL,byte 0FFh
735 000004AA 7501                JNZ $+3h
736 000004AC 40                 INC    AX
737 000004AD 59                 POP  CX
738 000004AE 20C8                AND       AL,CL
739 000004B0 50                 PUSH AX
740 000004B1 803E[8008]0D          CMP    [CHAR],byte 0Dh
741 000004B6 B0FF                MOV       AL,byte 0FFh
742 000004B8 7501                JNZ $+3h
743 000004BA 40                 INC    AX
744 000004BB 59                 POP   CX
745 000004BC 20C8                AND       AL,CL
746 000004BE D0D8                RCR       AL,1
747 000004C0 7333                JNB a@15
748                      ;do
749                      ;T=SIO$GET$BYTE;
750 000004C2 E8C9FD              CALL SIOGETBYTE ; gets valid hex byte and returns in AL
751 000004C5 A2[8A08]             MOV       [T],AL
752                      ;MEMORY$ARG1 = T;
753 000004C8 A0[8A08]             MOV       AL,[T]
754 000004CB C41E[8E08]           LES       BX,[MEMORYARG1PTR]
755 000004CF 268807              MOV       [ES:BX],byte AL
756 000004D2 C41E[8E08]           LES BX,[MEMORYARG1PTR]
757 000004D6 268A07              MOV AL,[ES:BX] ; read back memory just written
758 000004D9 3A06[8A08]           CMP     AL,[T] ; compare
759 000004DD 7416                JZ   a@15
760 000004DF B002                MOV       AL,byte 02h
```

```
761 000004E1 50                 PUSH AX
762 000004E2 E802FF               CALL SIOBLANKS
763                         ;call SIO$OUT$STRING(@('SUBSTITUTE FAILED'),17);
764 000004E5 2E8D06[2200]         LEA     AX,CS:@@LONG$CONSTANT$0013H;
765 000004EA 0E                 PUSH CS
766 000004EB 50                 PUSH AX
767 000004EC B011                 MOV         AL,byte 11h
768 000004EE 50                 PUSH AX
769 000004EF E8EFFC               CALL SIOOUTSTRING
770 000004F2 E927FC               JMP         ERROR
771                         a@15:
772                         ;ARG1.OFF = ARG1.OFF+1;
773 000004F5 8306[8E08]01         ADD     [ARG1],word 0001h;
774 000004FA E96FFF               JMP         a@50
775                         a@51:
776 000004FD 5D                  POP  BP
777 000004FE C3                 RET
778                         ;SIOEXAMMEM       ENDP
779
780
781                         SIODISPLAY:
782                         ;Implements the display byte command
783 000004FF 55                 PUSH BP
784 00000500 89E5                 MOV         BP,SP
785                         ;declare COUNT WORD;
786                         a@11:
787 00000502 E80AFD               CALL SIOGETCHAR
788                         ;do while CHAR=ASBL;
789 00000505 803E[8008]20         CMP     [CHAR],byte 20h
790 0000050A 74F6                 JZ    a@11
791                         ;call SIO$GET$ADDR(0);
792 0000050C B80000               MOV       AX,word 0000h
793 0000050F 50                 PUSH AX
794 00000510 E8B2FE               CALL SIOGETADDR
795                         ;if CHAR=ASCR then
796 00000513 803E[8008]0D         CMP     [CHAR],byte 0Dh
797 00000518 7508                 JNZ a@17
798                         ;count = 1;
799 0000051A C706[8408]0100       MOV [COUNT],word 0001h
800 00000520 EB1D                 JMP         a@18
801                         a@17:
802                         ;else
803                         ;do
804                         ;if CHAR=':' then goto ERROR;
```

```
805 00000522 803E[8008]3A          CMP    [CHAR],byte 3Ah
806 00000527 7503                JNZ $+5h
807 00000529 E9F0FB               JMP         ERROR
808 0000052C E8E0FC               CALL SIOGETCHAR
809                         ;COUNT = SIO$GET$WORD;
810 0000052F E8CBFD               CALL SIOGETWORD
811 00000532 A3[8408]             MOV       [COUNT],AX
812                         ;if CHAR<>ASCR then goto error
813 00000535 803E[8008]0D          CMP    [CHAR],byte 0Dh
814 0000053A 7403                JZ    $+5h
815 0000053C E9DDFB               JMP         ERROR
816                         ;end
817                         a@18:
818 0000053F E8D7FE               CALL SIONEWLINE
819                         ;call SIO$OUT$WORD(ARG1.OFF);
820 00000542 A1[8E08]             MOV       AX,[ARG1]
821 00000545 50                 PUSH AX
822 00000546 E881FC               CALL SIOOUTWORD ; write address
823 00000549 B003                MOV         AL,byte 03h
824 0000054B 50                 PUSH AX
825 0000054C E898FE               CALL SIOBLANKS
826                         ;if COUNT>1 then
827 0000054F 833E[8408]01          CMP    [COUNT],word 0001h
828 00000554 760F                JBE  a@54
829                         ;call SIO$BLANKS(LOW(3*(ARG1.OFF and 000Fh)));
830 00000556 A1[8E08]             MOV       AX,[ARG1]
831 00000559 83E00F               AND         AX,word 000Fh
832 0000055C B90300               MOV       CX,word 0003h
833 0000055F F7E1                MUL          CX
834                         a@12:
835 00000561 50                 PUSH AX
836 00000562 E882FE               CALL SIOBLANKS
837                         ;do while COUNT > 0;
838                         a@54:
839 00000565 833E[8408]00          CMP    [COUNT],word 0000h
840 0000056A 764B                JBE a@55
841 0000056C C41E[8E08]            LES       BX,[MEMORYARG1PTR]
842 00000570 26FF37               PUSH word [ES:BX]
843 00000573 E829FC               CALL SIOOUTBYTE
844                         ;call SIO$OUT$CHAR(' ');
845 00000576 B020                MOV         AL,byte 20h
846 00000578 50                 PUSH AX
847 00000579 E811FC               CALL SIOOUTCHAR
848                         ;ARG1.OFF = ARG1.OFF + 1;
```

```
849 0000057C 8306[8E08]01          ADD    [ARG1],word 0001h
850                        ;COUNT = COUNT - 1;
851 00000581 A1[8408]              MOV      AX,[COUNT]
852 00000584 83E801                SUB      AX,word 1h
853 00000587 A3[8408]              MOV      [COUNT],AX
854                        ;if COUNT > 0 and (ARG1.OFF and 000Fh) = 0 then
855 0000058A 09C0                  OR  AX,AX
856 0000058C B0FF                  MOV      AL,byte 0FFh
857 0000058E 7701                  JA   $+3h
858 00000590 40                   INC   AX
859 00000591 8B0E[8E08]             MOV    CX,[ARG1];
860 00000595 83E10F                AND      CX,word 0Fh
861 00000598 50                   PUSH AX
862 00000599 09C9                   OR  CX,CX
863 0000059B B0FF                   MOV      AL,byte 0FFh
864 0000059D 7401                   JZ   $+3h
865 0000059F 40                   INC AX
866 000005A0 59                   POP   CX
867 000005A1 20C8                   AND      AL,CL
868 000005A3 D0D8                    RCR     AL,1
869 000005A5 73BE                   JNB a@54
870                        ;do
871 000005A7 E86FFE                 CALL     SIONEWLINE
872                        ;call SIO$OUT$WORD(ARG1.OFF);
873 000005AA FF36[8E08]             PUSH word [ARG1]
874 000005AE E819FC                 CALL SIOOUTWORD
875                        ;call SIO$BLANKS(3);
876 000005B1 B003                   MOV      AL,byte 03h
877 000005B3 B400                   MOV      AH,byte 00h
878 000005B5 EBAA                   JMP      a@12
879                        ;end do
880                        a@55:
881 000005B7 5D                    POP  BP
882 000005B8 C3                    RET
883                        ;SIODISPLAY ENDP
884
885                        ;Command Dispatch Module
886
887                        GETCMND:
888                        ;Reads 1 character from the input stream
889                        ;and scans the command table. If a match is
890                        ;found, the index is returned as the value of
891                        ;the procedure. If a match is not found, the routine
892                        ;does not return, but jumps to the error routine.
```

```
  893 000005B9 55                PUSHBP
  894 000005BA 89E5               MOV     BP,SP
  895                     ;procedure BYTE;
  896                     ;declare I BYTE;
  897 000005BC E849FE             CALL    SIOCRLF
  898                     ;call SIO$OUT$CHAR('.');
  899 000005BF B02E               MOV     AL,byte 2Eh
  900 000005C1 50                PUSH AX
  901 000005C2 E8C8FB             CALL    SIOOUTCHAR
  902 000005C5 E847FC             CALL    SIOGETCHAR
  903                     ;do I=0 to 2;
  904 000005C8 C606[8308]00       MOV     [I],byte 0h
  905                     a@56:
  906 000005CD 803E[8308]02       CMP     [I],byte 2h
  907 000005D2 7603              JBE $+5h
  908 000005D4 E945FB             JMP     ERROR
  909                     ;if CHAR=SIO$CMND(I) then return I;
  910 000005D7 A0[8008]           MOV     AL,[CHAR]
  911 000005DA 8A1E[8308]         MOV     BL,[I]
  912 000005DE B700               MOV     BH,byte 0h
  913 000005E0 2E3A87[1000]       CMP     AL,[CS:SIOCMND+BX];
  914 000005E5 7505              JNZ a@23
  915 000005E7 A0[8308]           MOV     AL,[I]
  916 000005EA 5D                POP  BP
  917 000005EB C3                RET
  918                     a@23:
  919                     ;end do
  920 000005EC 8006[8308]01       ADD     [I],byte 1h
  921 000005F1 75DA              JNZ a@56
  922 000005F3 E926FB             JMP     ERROR
  923                     ;GETCMND   ENDP
  924
  925 000005F6 <res Ah>           TIMES 0600h-($-$$) resb 1; put ISRs starting at 0600h offset
  925       ******************    warning: uninitialized space declared in .text section:
zeroing [-w+zeroing]
  926                     ;interrupt service routines
  927                     DIVIDE$ERROR:
  928                     ; Prints divide error message and returns control to the monitor
  929 00000600 06                PUSH ES
  930 00000601 1E                PUSH DS
  931 00000602 2E8E1E[3700]       MOV DS,CS:@@DATA$FRAME
  932 00000607 50                PUSH AX
  933 00000608 51                PUSH CX
  934 00000609 52                PUSH DX
```

```
  935 0000060A 53                PUSH BX
  936 0000060B 56                PUSH SI
  937 0000060C 57                PUSH DI
  938 0000060D E80900              CALL DIVIDEERROR
  939 00000610 5F                POP DI
  940 00000611 5E                POP SI
  941 00000612 5B                POP BX
  942 00000613 5A                POP DX
  943 00000614 59                POP CX
  944 00000615 58                POP AX
  945 00000616 1F                POP DS
  946 00000617 07                POP ES
  947 00000618 CF                IRET
  948                    DIVIDEERROR:
  949 00000619 55                PUSH BP
  950 0000061A 89E5                MOV BP,SP
  951 0000061C 2E8D06[F607]          LEA AX,CS:@@LONG$CONSTANT$0024H
  952 00000621 0E                PUSH CS
  953 00000622 50                PUSH AX
  954 00000623 B00C                MOV AL,0ch
  955 00000625 50                PUSH AX
  956 00000626 E8B8FB              CALL SIOOUTSTRING
  957 00000629 E9B8FA              JMP NEXTCOMMAND ; check that BP does not need to be
POPPED
  958                    ;DIVIDEERROR ENDP
  959
  960                    NMI$:
  961                    ;prints a message that a non-maskable interrupt was received
  962 0000062C 06                PUSH ES
  963 0000062D 1E                PUSH DS
  964 0000062E 2E8E1E[3700]          MOV DS,CS:@@DATA$FRAME
  965 00000633 50                PUSH AX
  966 00000634 51                PUSH CX
  967 00000635 52                PUSH DX
  968 00000636 53                PUSH BX
  969 00000637 56                PUSH SI
  970 00000638 57                PUSH DI
  971 00000639 E80900              CALL NMI
  972 0000063C 5F                POP DI
  973 0000063D 5E                POP SI
  974 0000063E 5B                POP BX
  975 0000063F 5A                POP DX
  976 00000640 59                POP CX
  977 00000641 58                POP AX
```

```
 978 00000642 1F                POP DS
 979 00000643 07                POP ES
 980 00000644 CF                IRET
 981                    NMI:
 982 00000645 55                PUSH BP
 983 00000646 89E5                MOV BP,SP
 984 00000648 2E8D06[0208]          LEA AX,CS:@@LONG$CONSTANT$0030H
 985 0000064D 0E                PUSH CS
 986 0000064E 50                PUSH AX
 987 0000064F B011                MOV AL,11h
 988 00000651 50                PUSH AX
 989 00000652 E88CFB              CALL SIOOUTSTRING
 990 00000655 E8C1FD              CALL SIONEWLINE
 991 00000658 5D                POP BP
 992 00000659 C3                RET
 993                    ;NMI ENDP
 994
 995                    OVEFLOW$ERROR:
 996                    ; prints overflow error message and returns control to monitor
 997 0000065A 06                PUSH ES
 998 0000065B 1E                PUSH DS
 999 0000065C 2E8E1E[3700]          MOV DS,CS:@@DATA$FRAME
1000 00000661 50                PUSH AX
1001 00000662 51                PUSH CX
1002 00000663 52                PUSH DX
1003 00000664 53                PUSH BX
1004 00000665 56                PUSH SI
1005 00000666 57                PUSH DI
1006 00000667 E80900              CALL OVERFLOWERROR
1007 0000066A 5F                POP DI
1008 0000066B 5E                POP SI
1009 0000066C 5B                POP BX
1010 0000066D 5A                POP DX
1011 0000066E 59                POP CX
1012 0000066F 58                POP AX
1013 00000670 1F                POP DS
1014 00000671 07                POP ES
1015 00000672 CF                IRET
1016                     OVERFLOWERROR:
1017 00000673 55                PUSH BP
1018 00000674 89E5                MOV BP,SP
1019 00000676 2E8D06[1308]          LEA AX,CS:@@LONG$CONSTANT$0041H
1020 0000067B 0E                PUSH CS
1021 0000067C 50                PUSH AX
```

```
1022 0000067D B008              MOV AL,08h
1023 0000067F 50                PUSH AX
1024 00000680 E85EFB            CALL SIOOUTSTRING
1025 00000683 E95EFA            JMP NEXTCOMMAND ; check that BP does not need to be
POPPED
1026                    ;OVERFLOWERROR ENDP
1027
1028                    ;  The following 8 ISRs correspond to interrupts from the 8259A
1029
1030                    INT32$: ;Procedure interrupt 32
1031 00000686 06               PUSH ES
1032 00000687 1E               PUSH DS
1033 00000688 2E8E1E[3700]          MOV DS,CS:@@DATA$FRAME
1034 0000068D 50               PUSH AX
1035 0000068E 51               PUSH CX
1036 0000068F 52               PUSH DX
1037 00000690 53               PUSH BX
1038 00000691 56               PUSH SI
1039 00000692 57               PUSH DI
1040 00000693 E80900            CALL INT32
1041 00000696 5F               POP DI
1042 00000697 5E               POP SI
1043 00000698 5B               POP BX
1044 00000699 5A               POP DX
1045 0000069A 59               POP CX
1046 0000069B 58               POP AX
1047 0000069C 1F               POP DS
1048 0000069D 07               POP ES
1049 0000069E CF               IRET
1050                    INT32:
1051 0000069F 55               PUSH BP
1052 000006A0 89E5              MOV BP,SP
1053 000006A2 2E8D06[1B08]          LEA AX,CS:@@LONG$CONSTANT$0049H
1054 000006A7 0E               PUSH CS
1055 000006A8 50               PUSH AX
1056 000006A9 B00C              MOV AL,0ch
1057 000006AB 50               PUSH AX
1058 000006AC E832FB            CALL SIOOUTSTRING
1059 000006AF E867FD            CALL SIONEWLINE
1060 000006B2 5D               POP BP
1061 000006B3 C3               RET
1062                    ;INT32 ENDP
1063
1064                    INT33$: ;Procedure interrupt 33
```

```
1065 000006B4 06                PUSH ES
1066 000006B5 1E                PUSH DS
1067 000006B6 2E8E1E[3700]          MOV DS,CS:@@DATA$FRAME
1068 000006BB 50                PUSH AX
1069 000006BC 51                PUSH CX
1070 000006BD 52                PUSH DX
1071 000006BE 53                PUSH BX
1072 000006BF 56                PUSH SI
1073 000006C0 57                PUSH DI
1074 000006C1 E80900              CALL INT33
1075 000006C4 5F                POP DI
1076 000006C5 5E                POP SI
1077 000006C6 5B                POP BX
1078 000006C7 5A                POP DX
1079 000006C8 59                POP CX
1080 000006C9 58                POP AX
1081 000006CA 1F                POP DS
1082 000006CB 07                POP ES
1083 000006CC CF                IRET
1084                        INT33:
1085 000006CD 55                PUSH BP
1086 000006CE 89E5                MOV BP,SP
1087 000006D0 2E8D06[2708]          LEA AX,CS:@@LONG$CONSTANT$0055H
1088 000006D5 0E                PUSH CS
1089 000006D6 50                PUSH AX
1090 000006D7 B00C                MOV AL,0ch
1091 000006D9 50                PUSH AX
1092 000006DA E804FB              CALL SIOOUTSTRING
1093 000006DD E839FD              CALL SIONEWLINE
1094 000006E0 5D                POP BP
1095 000006E1 C3                RET
1096                        ;INT33 ENDP
1097
1098                        INT34$: ;Procedure interrupt 34
1099 000006E2 06                PUSH ES
1100 000006E3 1E                PUSH DS
1101 000006E4 2E8E1E[3700]          MOV DS,CS:@@DATA$FRAME
1102 000006E9 50                PUSH AX
1103 000006EA 51                PUSH CX
1104 000006EB 52                PUSH DX
1105 000006EC 53                PUSH BX
1106 000006ED 56                PUSH SI
1107 000006EE 57                PUSH DI
1108 000006EF E80900              CALL INT34
```

```
1109 000006F2 5F                POP DI
1110 000006F3 5E                POP SI
1111 000006F4 5B                POP BX
1112 000006F5 5A                POP DX
1113 000006F6 59                POP CX
1114 000006F7 58                POP AX
1115 000006F8 1F                POP DS
1116 000006F9 07                POP ES
1117 000006FA CF                IRET
1118                    INT34:
1119 000006FB 55                PUSH BP
1120 000006FC 89E5                MOV BP,SP
1121 000006FE 2E8D06[3308]          LEA AX,CS:@@LONG$CONSTANT$0061H
1122 00000703 0E                PUSH CS
1123 00000704 50                PUSH AX
1124 00000705 B00C                MOV AL,0ch
1125 00000707 50                PUSH AX
1126 00000708 E8D6FA              CALL SIOOUTSTRING
1127 0000070B E80BFD              CALL SIONEWLINE
1128 0000070E 5D                POP BP
1129 0000070F C3                RET
1130                    ;INT34 ENDP
1131
1132                    INT35$: ;Procedure interrupt 35
1133 00000710 06                PUSH ES
1134 00000711 1E                PUSH DS
1135 00000712 2E8E1E[3700]          MOV DS,CS:@@DATA$FRAME
1136 00000717 50                PUSH AX
1137 00000718 51                PUSH CX
1138 00000719 52                PUSH DX
1139 0000071A 53                PUSH BX
1140 0000071B 56                PUSH SI
1141 0000071C 57                PUSH DI
1142 0000071D E80900              CALL INT35
1143 00000720 5F                POP DI
1144 00000721 5E                POP SI
1145 00000722 5B                POP BX
1146 00000723 5A                POP DX
1147 00000724 59                POP CX
1148 00000725 58                POP AX
1149 00000726 1F                POP DS
1150 00000727 07                POP ES
1151 00000728 CF                IRET
1152                    INT35:
```

```
1153 00000729 55                PUSH BP
1154 0000072A 89E5                MOV BP,SP
1155 0000072C 2E8D06[3F08]          LEA AX,CS:@@LONG$CONSTANT$006DH
1156 00000731 0E                PUSH CS
1157 00000732 50                PUSH AX
1158 00000733 B00C                MOV AL,0ch
1159 00000735 50                PUSH AX
1160 00000736 E8A8FA              CALL SIOOUTSTRING
1161 00000739 E8DDFC              CALL SIONEWLINE
1162 0000073C 5D                POP BP
1163 0000073D C3                RET
1164                        ;INT35 ENDP
1165
1166                        INT36$: ;Procedure interrupt 36
1167 0000073E 06                PUSH ES
1168 0000073F 1E                PUSH DS
1169 00000740 2E8E1E[3700]          MOV DS,CS:@@DATA$FRAME
1170 00000745 50                PUSH AX
1171 00000746 51                PUSH CX
1172 00000747 52                PUSH DX
1173 00000748 53                PUSH BX
1174 00000749 56                PUSH SI
1175 0000074A 57                PUSH DI
1176 0000074B E80900              CALL INT36
1177 0000074E 5F                POP DI
1178 0000074F 5E                POP SI
1179 00000750 5B                POP BX
1180 00000751 5A                POP DX
1181 00000752 59                POP CX
1182 00000753 58                POP AX
1183 00000754 1F                POP DS
1184 00000755 07                POP ES
1185 00000756 CF                IRET
1186                        INT36:
1187 00000757 55                PUSH BP
1188 00000758 89E5                MOV BP,SP
1189 0000075A 2E8D06[4B08]          LEA AX,CS:@@LONG$CONSTANT$0079H
1190 0000075F 0E                PUSH CS
1191 00000760 50                PUSH AX
1192 00000761 B00C                MOV AL,0ch
1193 00000763 50                PUSH AX
1194 00000764 E87AFA              CALL SIOOUTSTRING
1195 00000767 E8AFFC              CALL SIONEWLINE
1196 0000076A 5D                POP BP
```

```
1197 0000076B C3                RET
1198                    ;INT36 ENDP
1199
1200                    INT37$: ;Procedure interrupt 37
1201 0000076C 06                PUSH ES
1202 0000076D 1E                PUSH DS
1203 0000076E 2E8E1E[3700]          MOV DS,CS:@@DATA$FRAME
1204 00000773 50                PUSH AX
1205 00000774 51                PUSH CX
1206 00000775 52                PUSH DX
1207 00000776 53                PUSH BX
1208 00000777 56                PUSH SI
1209 00000778 57                PUSH DI
1210 00000779 E80900              CALL INT37
1211 0000077C 5F                POP DI
1212 0000077D 5E                POP SI
1213 0000077E 5B                POP BX
1214 0000077F 5A                POP DX
1215 00000780 59                POP CX
1216 00000781 58                POP AX
1217 00000782 1F                POP DS
1218 00000783 07                POP ES
1219 00000784 CF                IRET
1220                    INT37:
1221 00000785 55                PUSH BP
1222 00000786 89E5                MOV BP,SP
1223 00000788 2E8D06[5708]          LEA AX,CS:@@LONG$CONSTANT$0085H
1224 0000078D 0E                PUSH CS
1225 0000078E 50                PUSH AX
1226 0000078F B00C                MOV AL,0ch
1227 00000791 50                PUSH AX
1228 00000792 E84CFA              CALL SIOOUTSTRING
1229 00000795 E881FC              CALL SIONEWLINE
1230 00000798 5D                POP BP
1231 00000799 C3                RET
1232                    ;INT37 ENDP
1233
1234                    INT38$: ;Procedure interrupt 38
1235 0000079A 06                PUSH ES
1236 0000079B 1E                PUSH DS
1237 0000079C 2E8E1E[3700]          MOV DS,CS:@@DATA$FRAME
1238 000007A1 50                PUSH AX
1239 000007A2 51                PUSH CX
1240 000007A3 52                PUSH DX
```

```
1241 000007A4 53                PUSH BX
1242 000007A5 56                PUSH SI
1243 000007A6 57                PUSH DI
1244 000007A7 E80900              CALL INT38
1245 000007AA 5F                POP DI
1246 000007AB 5E                POP SI
1247 000007AC 5B                POP BX
1248 000007AD 5A                POP DX
1249 000007AE 59                POP CX
1250 000007AF 58                POP AX
1251 000007B0 1F                POP DS
1252 000007B1 07                POP ES
1253 000007B2 CF                IRET
1254                    INT38:
1255 000007B3 55                PUSH BP
1256 000007B4 89E5                MOV BP,SP
1257 000007B6 2E8D06[6308]          LEA AX,CS:@@LONG$CONSTANT$0091H
1258 000007BB 0E                PUSH CS
1259 000007BC 50                PUSH AX
1260 000007BD B00C                MOV AL,0ch
1261 000007BF 50                PUSH AX
1262 000007C0 E81EFA              CALL SIOOUTSTRING
1263 000007C3 E853FC              CALL SIONEWLINE
1264 000007C6 5D                POP BP
1265 000007C7 C3                RET
1266                    ;INT38 ENDP
1267
1268                    INT39$: ;Procedure interrupt 39
1269 000007C8 06                PUSH ES
1270 000007C9 1E                PUSH DS
1271 000007CA 2E8E1E[3700]          MOV DS,CS:@@DATA$FRAME
1272 000007CF 50                PUSH AX
1273 000007D0 51                PUSH CX
1274 000007D1 52                PUSH DX
1275 000007D2 53                PUSH BX
1276 000007D3 56                PUSH SI
1277 000007D4 57                PUSH DI
1278 000007D5 E80900              CALL INT39
1279 000007D8 5F                POP DI
1280 000007D9 5E                POP SI
1281 000007DA 5B                POP BX
1282 000007DB 5A                POP DX
1283 000007DC 59                POP CX
1284 000007DD 58                POP AX
```

```
 1285 000007DE 1F                  POP DS
 1286 000007DF 07                  POP ES
 1287 000007E0 CF                  IRET
 1288                        INT39:
 1289 000007E1 55                  PUSH BP
 1290 000007E2 89E5                  MOV BP,SP
 1291 000007E4 2E8D06[6F08]          LEA AX,CS:@@LONG$CONSTANT$009DH
 1292 000007E9 0E                  PUSH CS
 1293 000007EA 50                  PUSH AX
 1294 000007EB B00C                  MOV AL,0ch
 1295 000007ED 50                  PUSH AX
 1296 000007EE E8F0F9                CALL SIOOUTSTRING
 1297 000007F1 E825FC                CALL SIONEWLINE
 1298 000007F4 5D                  POP BP
 1299 000007F5 C3                  RET
 1300                        ;INT39 ENDP
 1301
 1302
 1303 000007F6 444956494445204552-    @@LONG$CONSTANT$0024H db      'DIVIDE
ERROR'
 1303 000007FF 524F52
 1304 00000802 4E4F4E204D41534B41-    @@LONG$CONSTANT$0030H db      'NON
MASKABLE INTR'
 1304 0000080B 424C4520494E5452
 1305 00000813 4F564552464C4F57      @@LONG$CONSTANT$0041H   db      'OVERFLOW'
 1306 0000081B 494E54455252555054-    @@LONG$CONSTANT$0049H db      'INTERRUPT
32'
 1306 00000824 203332
 1307 00000827 494E54455252555054-    @@LONG$CONSTANT$0055H db      'INTERRUPT
33'
 1307 00000830 203333
 1308 00000833 494E54455252555054-    @@LONG$CONSTANT$0061H db      'INTERRUPT
34'
 1308 0000083C 203334
 1309 0000083F 494E54455252555054-    @@LONG$CONSTANT$006DH db      'INTERRUPT
35'
 1309 00000848 203335
 1310 0000084B 494E54455252555054-    @@LONG$CONSTANT$0079H db      'INTERRUPT
36'
 1310 00000854 203336
 1311 00000857 494E54455252555054-    @@LONG$CONSTANT$0085H db      'INTERRUPT
37'
 1311 00000860 203337
```

```
 1312 00000863 494E54455252555054-     @@LONG$CONSTANT$0091H db       'INTERRUPT
38'
 1312 0000086C 203338
 1313 0000086F 494E54455252555054-     @@LONG$CONSTANT$009DH db       'INTERRUPT
39'
 1313 00000878 203339
 1314
 1315
 1316                          ; manually storing the bootstrap instruction in EEPROM at the correct
physical address
 1317                          ; so don't really need it in the binary file output.  Just adds a lot of
zeros before the few bytes
 1318                          ; of instruction.
 1319                          ; Boot Strap Procedure
 1320                          ;8086 reset sets program counter to 0FFFF0h
 1321                          ;bootstrap:
 1322                          ;TIMES 09F0h-($-$$) DB 0 ; makes offset relative to code segment -
 1323                          ; address 0FFFF0h-code segment address of 0FF60h (0FF600) = 09F0h
 1324                          ; note the addresses here are relative to the code segment start, and
are not physical addresses.  If
 1325                          ; writing this boot instruction to memory it needs to be written to
0xff0.
 1326                          ;db  0EAh  ; JMP
 1327                          ;dw  00BCh ; Offset
 1328                          ;dw  0FF60h; Code segment (shifted left 4 bits before use)
 1329                          ;end 8086TestUart4;
 1330
 1331                          ; RAM variable storage
 1332                          ;section .bss
 1333                          ; 07d2 did not fix after code unless I figure out how to specify different
segments
 1334                          ;TIMES 07d2h-($-$$) resb 1; store after stack in same segment/frame
 1335 0000087B ??????????            TIMES 0880h-($-$$) resb 1; store after stack in same
segment/frame
 1335          ******************     warning: uninitialized space declared in .text section:
zeroing [-w+zeroing]
 1336
 1337 00000880 00                    CHAR                db      00h
 1338 00000881 00                    C                   db      00h
 1339 00000882 00                    B                   db      00h
 1340 00000883 00                    I                   db      00h
 1341 00000884 00                    COUNT               db      00h
 1342 00000885 00                    H                   db      00h
 1343 00000886 00                    N                   db      00h
```

```
1344 00000887 00              OPER                db      00h
1345 00000888 0000              SAVE              dw      0000h
1346 0000088A 0000              T                 dw      0000h
1347 0000088C 0000              W                 dw      0000h
1348
1349                    ;Memory Argument Section
1350                    MEMORYARG1PTR:
1351                    ARG1:
1352 0000088E 0000              OFFSET:           dw      0000h
1353 00000890 0000              MYSEG:            dw      0000h
```

Appendix C. EEPROM Writing Code

```
/*
 * main.c
 *
 * Created: 11/28/2021 3:49:47 PM
 * modified to write data to EEPROM 1/13/2021
 *  Author: john
 *  Adding interrupt vector table write 2/24/2022
 */

#include <asf.h>
#include <stdio.h>
#include "cpuspeed.h"
#include <util/delay.h> //#include "delay.h"
#include "uart.h"

int main(void)
{
uint16_t i,ierror_count,iPass,iMaxAddress,iAStart,iLowHigh,iOutAddr;
uint8_t value, value2;
uint8_t iBootStrap[5] = {0xea,0xbc,0x00,0x60,0xff};
uint8_t iIVT[64] = {0x00,0x06,0x60,0xff,0x86,0x06,0x60,0xff,
       0x2c,0x06,0x60,0xff,0x86,0x06,0x60,0xff,
       0x5a,0x06,0x60,0xff,0x86,0x06,0x60,0xff,
       0x86,0x06,0x60,0xff,0x86,0x06,0x60,0xff,
       0x86,0x06,0x60,0xff,0xb4,0x06,0x60,0xff,
       0xe2,0x06,0x60,0xff,0x10,0x07,0x60,0xff,
       0x3e,0x07,0x60,0xff,0x6c,0x07,0x60,0xff,
       0x9a,0x07,0x60,0xff,0xc8,0x07,0x60,0xff};

// ROM data size = 1520 (5F0h) + space for boot code
#include "ROMDataUart4.h"

       /* Insert system clock initialization code here (sysclk_init()). */
       // Part of code below is to make sure EEPROM is not accidentally written
//     DDRA = 0xff;
//     DDRB = 0x1f;
       DDRB = 0x0f;
       DDRD = DDRD|0x30;

//     PORTA = 0x00; // start reading from address 0x0000 (address is 11 bits)
//     PORTB = 0x18; // chip select set off, may need or else there will be 1 error?
       PORTB = 0x08; // chip select set off, may need or else there will be 1 error?
//     PORTD |= (1<<4); //write off and OE inactive (both high)
       PORTD |= (3<<4); //write off and OE inactive (both high)

       board_init();

       _delay_ms(50);

// start up the serial port
       uart_init(9600);
       FILE uart_stream = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);
       stdin = stdout = &uart_stream;

       //write out input data stored in romdata
/*     for (i=0;i<0x450;i++) {
```

```
                printf_P(PSTR("%x: value = %4x\r\n"),i,romdata[i]);
        } */

// USART0 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART0 Receiver: On
// USART0 Transmitter: On
// USART0 Mode: Asynchronous
// USART0 Baud Rate: 9600
/*UCSR0A=0x00;
UCSR0B=0x18;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x2F; */

//Write address lines to port A 0-7 and B0-2
// Write chip enable ~CE and to B3
DDRA = 0xff;  // Port A address pins are output
DDRB = 0x1f; // Upper address output on bits 0-2 and ~CE on bit 3
DDRC = 0xff; // 8 bits write - set port C pins output for write
// Port D pins 4 and 5 control
DDRD = DDRD|0x30; //D4 is ~WE and D5 is ~OE

PORTC = 0x00; // start out output set to zero
//_delay_ms(50);

// Note: calls to _delay_ms and/or _delay_us appear to cause problems in below loops

//EEPROM write when low pulse on ~WE D4 and ~OE is high
// loop to write data
// absolute EEPROM starting address = FF000h
// Base starting address of code segment = FF600h (=CS of FF60x10)
// Address local to low or high byte ROM is only from A1 up so shift right 1 bit
// Get 0xB00, which only using A1 to A11, becomes 0x300, Max Address used = 0x750

iMaxAddress = 0x880; //maximum data to write
iAStart = 0x300;
iLowHigh = 1; // 0=write low bytes, 1=write high bytes

iOutAddr = iAStart;
for(i=iLowHigh; i<iMaxAddress; i+=2) { // i is index to source data
        value = romdata[i];
// Start with ~OE low and ~CE high
        PORTD &= ~(1<<5); //Output enable ~OE low
        PORTB |= (1<<3); //~CE high
        PORTD |= (1<<4); //write line high
        PORTA = (iOutAddr&0xff); // set low address bits
        PORTB &= ~(0x07); // clear lower 3 bits of B
        PORTB |= ((0x700&iOutAddr)>>8); // high address bits
// Set ~OE high and ~CE low
        PORTD |= (1<<5); //~OE high
        PORTB &= ~(1<<3);  // ~CE Low
        PORTC = value;
        PORTD &= ~(1<<4); // ~WE write line low, initiate write address read (~OE must be
high)
        PORTD |= (1<<4); // ~WE write line high
        // Note: need delay or print here or else first value won't be written correctly
        _delay_ms(25);
```

```
        printf_P(PSTR("W: %.4x %.2x \r\n"),iOutAddr,value);
        iOutAddr++;
}

// write bootstrap code
//iAStart = 0x9f0;
iAStart = 0x7f8;
iOutAddr = iAStart;
for(i=iLowHigh; i<5; i+=2) { // i is index to source data
        value = iBootStrap[i];
        // Start with ~OE low and ~CE high
        PORTD &= ~(1<<5); //Output enable ~OE low
        PORTB |= (1<<3); //~CE high
        PORTD |= (1<<4); //write line high
        PORTA = (iOutAddr&0xff); // set low address bits
        PORTB &= ~(0x07); // clear lower 3 bits of B
        PORTB |= ((0x700&iOutAddr)>>8); // high address bits
        // Set ~OE high and ~CE low
        PORTD |= (1<<5); //~OE high
        PORTB &= ~(1<<3);  // ~CE Low
        // Write to EEPROM
        PORTC = value;
        PORTD &= ~(1<<4); // ~WE write line low, initiate write address read (~OE must be
high)
        PORTD |= (1<<4); // ~WE write line high
        // Note: need delay or print here or else first value won't be written correctly
        //_delay_ms(25);
        printf_P(PSTR("B: %.4x %.2x \r\n"),iOutAddr,value);
        iOutAddr++;
}

// write interrupt vector table (IVT)
iAStart = 0x000;
iOutAddr = iAStart;
for(i=iLowHigh; i<64; i+=2) { // i is index to source data
        value = iIVT[i];
        // Start with ~OE low and ~CE high
        PORTD &= ~(1<<5); //Output enable ~OE low
        PORTB |= (1<<3); //~CE high
        PORTD |= (1<<4); //write line high
        PORTA = (iOutAddr&0xff); // set low address bits
        PORTB &= ~(0x07); // clear lower 3 bits of B
        PORTB |= ((0x700&iOutAddr)>>8); // high address bits
        // Set ~OE high and ~CE low
        PORTD |= (1<<5); //~OE high
        PORTB &= ~(1<<3);  // ~CE Low
        // Write to EEPROM
        PORTC = value;
        PORTD &= ~(1<<4); // ~WE write line low, initiate write address read (~OE must be
high)
        PORTD |= (1<<4); // ~WE write line high
        // Note: need delay or print here or else first value won't be written correctly
        //_delay_ms(25);
        printf_P(PSTR("IVT: %.4x %.2x \r\n"),iOutAddr,value);
        iOutAddr++;
}
```

```
//Read back EEPROM data
DDRC = 0x00; // all 8 bits read
value2 = PINC; // clear any buffering
ierror_count = 0;
iPass = 0;

// Read loop ~CE(B3) and ~OE (D5) are low and ~WE (D4) is high
//Start with ~CE and ~OE High
iAStart = 0x300;
iOutAddr = iAStart;
for(i=iLowHigh; i<iMaxAddress; i+=2) {
       value = romdata[i];  //expected to read
       PORTD |= (1<<5); //~OE high
       PORTB |= (1<<3); //~CE high
       PORTA = (iOutAddr&0xff); // set low address bits
       PORTB &= ~(0x07); // clear lower 3 bits of B
       PORTB |= ((0x700&iOutAddr)>>8); //set upper address bits
       PORTD &= ~(1<<5); //Output enable ~OE low (read address)
       PORTB &= ~(1<<3); //Output enable ~CE low (read address)
       // read back from chip
       PORTD |= (1<<5); //~OE high (Make output of EEPROM valid)
       PORTB |= (1<<3); //~CE high (Make output of EEPROM valid)
       value2 = PINC; // read value
//     _delay_ms(50);
printf_P(PSTR("%.4x %.2x %.2x\r\n"),iOutAddr,value,value2);
       if (value != value2) ierror_count++;
       iOutAddr++;
} // end for

printf_P(PSTR("%d: Error Count = %4d\r\n"),iPass,ierror_count);

iAStart = 0x7f8;
ierror_count = 0;
// read bootstrap code
// Read loop ~CE(B3) and ~OE (D5) are low and ~WE (D4) is high
//Start with ~CE and ~OE High
iOutAddr = iAStart;
for(i=iLowHigh; i<5; i+=2) {
       value = iBootStrap[i];  //expected to read
       PORTD |= (1<<5); //~OE high
       PORTB |= (1<<3); //~CE high
       PORTA = (iOutAddr&0xff); // set low address bits
       PORTB &= ~(0x07); // clear lower 3 bits of B
       PORTB |= ((0x700&iOutAddr)>>8); //set upper address bits
       PORTD &= ~(1<<5); //Output enable ~OE low (read address)
       PORTB &= ~(1<<3); //Output enable ~CE low (read address)
       // read back from chip
       PORTD |= (1<<5); //~OE high (Make output of EEPROM valid)
       PORTB |= (1<<3); //~CE high (Make output of EEPROM valid)
       value2 = PINC; // read value
       //     value2 = value2&0x0f;
       //     _delay_ms(50);
       printf_P(PSTR("B %.4x %.2x %.2x\r\n"),iOutAddr,value,value2);
       if (value != value2) ierror_count++;
       iOutAddr++;
}
printf_P(PSTR("%d: Error Count = %4d\r\n"),iPass,ierror_count);
```

```
iAStart = 0x000;
ierror_count = 0;
// read IVT
// Read loop ~CE(B3) and ~OE (D5) are low and ~WE (D4) is high
//Start with ~CE and ~OE High
iOutAddr = iAStart;
for(i=iLowHigh; i<64; i+=2) {
       value = iIVT[i];  //expected to read
       PORTD |= (1<<5); //~OE high
       PORTB |= (1<<3); //~CE high
       PORTA = (iOutAddr&0xff); // set low address bits
       PORTB &= ~(0x07); // clear lower 3 bits of B
       PORTB |= ((0x700&iOutAddr)>>8); //set upper address bits
       PORTD &= ~(1<<5); //Output enable ~OE low (read address)
       PORTB &= ~(1<<3); //Output enable ~CE low (read address)
       // read back from chip
       PORTD |= (1<<5); //~OE high (Make output of EEPROM valid)
       PORTB |= (1<<3); //~CE high (Make output of EEPROM valid)
       value2 = PINC; // read value
       //     value2 = value2&0x0f;
       //     _delay_ms(50);
       printf_P(PSTR("IVT %.4x %.2x %.2x\r\n"),iOutAddr,value,value2);
       if (value != value2) ierror_count++;
       iOutAddr++;
}
printf_P(PSTR("%d: Error Count = %4d\r\n"),iPass,ierror_count);

PORTB |= (1<<3);  // ~CE High (deselect)
//PORTA = 0x01; // reset address to zero
//PORTB = 0x18; // chip enable and read set high
PORTD &= ~(1<<5); // Set chip output enable low

while (1)
{
              _delay_ms(10); // was 1
}
}
```

```
uint8_t romdata[0x880]=
{0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x41,0x42,0x43,0x44,0x45,0x46,
       0x47,0x53,0x44,0x44,0x45,0x4d,0x4f,0x2d,0x38,0x36,0x20,0x56,0x45,0x52,0x20,0x31,
       0x2e,0x32,0x53,0x55,0x42,0x53,0x54,0x49,0x54,0x55,0x54,0x45,0x20,0x46,0x41,0x49,
       0x4c,0x45,0x44,0xd0,0x07,0x01,0x01,0x01,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfa,0xb8,0x01,0x01,
```

0x8e,0xd0,0x2e,0x8e,0x1e,0x37,0x00,0xfb,0xb8,0xd0,0x07,0x89,0xc4,0x89,0xe5,0xe8,
0x8d,0x00,0x2e,0x8d,0x06,0x13,0x00,0x0e,0x50,0xb0,0x0f,0x50,0xe8,0x03,0x01,0xe8,
0x38,0x03,0xe8,0x2b,0x01,0xb8,0xd0,0x07,0x89,0xc4,0x2e,0x8e,0x1e,0x37,0x00,0xb0,
0xff,0xd0,0xd8,0x73,0x28,0xb8,0xd0,0x07,0x89,0xc4,0xe8,0xbd,0x04,0xb4,0x00,0x89,
0xc3,0xd1,0xe3,0x2e,0xff,0xa7,0x17,0x01,0xe8,0x1d,0x03,0xeb,0xe2,0xe8,0x3f,0x03,
0xeb,0xdd,0xe8,0xeb,0x03,0xeb,0xd8,0x08,0x01,0x0d,0x01,0x12,0x01,0x2e,0x8e,0x1e,
0x37,0x00,0xb8,0xd0,0x07,0x89,0xc4,0xb0,0x2a,0x50,0xe8,0x61,0x00,0xeb,0xb6,0x55,
0x89,0xe5,0xb0,0x37,0xe6,0x00,0xb0,0x01,0x50,0xe8,0x10,0x00,0xb0,0x00,0xe6,0x02,
0xb0,0x01,0x50,0xe8,0x06,0x00,0xb0,0x01,0xe6,0x02,0x5d,0xc3,0x55,0x89,0xe5,0x80,
0x7e,0x04,0x00,0x76,0x06,0x80,0x6e,0x04,0x01,0xeb,0xf4,0x5d,0xc2,0x02,0x00,0x55,
0x89,0xe5,0xb4,0x00,0xb0,0x80,0xe6,0x26,0xb0,0x02,0x50,0xe8,0xde,0xff,0xb0,0x0c,
0xe6,0x20,0xb0,0x02,0x50,0xe8,0xd4,0xff,0xb0,0x00,0xe6,0x22,0xb0,0x02,0x50,0xe8,
0xca,0xff,0xb0,0x03,0xe6,0x26,0xb0,0x02,0x50,0xe8,0xc0,0xff,0x5d,0xc3,0x55,0x89,
0xe5,0xe4,0x2a,0xa8,0x20,0x74,0xfa,0x8a,0x46,0x04,0xe6,0x20,0x5d,0xc2,0x02,0x00,
0x55,0x89,0xe5,0x8a,0x5e,0x04,0xb1,0x04,0xd2,0xeb,0x80,0xe3,0x0f,0xb7,0x00,0x2e,
0xff,0xb7,0x00,0x00,0xe8,0xd7,0xff,0x8a,0x5e,0x04,0x80,0xe3,0x0f,0xb7,0x00,0x2e,
0xff,0xb7,0x00,0x00,0xe8,0xc7,0xff,0x5d,0xc2,0x02,0x00,0x55,0x89,0xe5,0x8b,0x46,
0x04,0x88,0xe0,0x50,0xe8,0xc9,0xff,0x8b,0x46,0x04,0x50,0xe8,0xc2,0xff,0x5d,0xc2,
0x02,0x00,0x55,0x89,0xe5,0xc6,0x06,0x83,0x08,0x00,0x8a,0x46,0x04,0x2c,0x01,0x38,
0x06,0x83,0x08,0x77,0x17,0xa0,0x83,0x08,0xb4,0x00,0x89,0xc6,0xc4,0x5e,0x06,0x26,
0xff,0x30,0xe8,0x89,0xff,0x80,0x06,0x83,0x08,0x01,0x75,0xde,0x5d,0xc2,0x06,0x00,
0x55,0x89,0xe5,0xe4,0x2a,0xa8,0x01,0x74,0xfa,0xe4,0x20,0x24,0x7f,0xa2,0x80,0x08,
0x3c,0x20,0x72,0x07,0xff,0x36,0x80,0x08,0xe8,0x63,0xff,0x5d,0xc3,0x55,0x89,0xe5,
0xb1,0x30,0x8a,0x46,0x04,0x38,0xc1,0xb0,0xff,0x76,0x01,0x40,0x8a,0x4e,0x04,0x50,
0x80,0xf9,0x39,0xb0,0xff,0x76,0x01,0x40,0x5a,0x20,0xd0,0xb2,0x41,0x50,0x38,0xca,
0xb0,0xff,0x76,0x01,0x40,0x50,0x80,0xf9,0x46,0xb0,0xff,0x76,0x01,0x40,0x59,0x20,
0xc8,0x59,0x08,0xc8,0xd0,0xd8,0x73,0x04,0xb0,0xff,0xeb,0x02,0xb0,0x00,0x5d,0xc2,
0x02,0x00,0x55,0x89,0xe5,0x8a,0x46,0x04,0x2c,0x30,0x88,0x46,0x04,0x3c,0x09,0x77,
0x05,0x8a,0x46,0x04,0xeb,0x05,0x8a,0x46,0x04,0x2c,0x07,0x5d,0xc2,0x02,0x00,0x55,
0x89,0xe5,0xff,0x36,0x80,0x08,0xe8,0x94,0xff,0xf6,0xd0,0xd0,0xd8,0x73,0x03,0xe9,
0x7b,0xfe,0xc6,0x06,0x82,0x08,0x00,0xff,0x36,0x80,0x08,0xe8,0x7f,0xff,0xd0,0xd8,
0x73,0x1a,0xa0,0x82,0x08,0xb1,0x04,0xd2,0xe0,0x50,0xff,0x36,0x80,0x08,0xe8,0xb1,
0xff,0x59,0x00,0xc8,0xa2,0x82,0x08,0xe8,0x46,0xff,0xeb,0xdb,0xa0,0x80,0x08,0x3c,
0x0d,0xb0,0xff,0x74,0x01,0x40,0x50,0x80,0x3e,0x80,0x08,0x20,0xb0,0xff,0x74,0x01,
0x40,0x59,0x08,0xc8,0x50,0x80,0x3e,0x80,0x08,0x2c,0xb0,0xff,0x74,0x01,0x40,0x59,
0x08,0xc8,0xd0,0xd8,0x72,0x03,0xe9,0x24,0xfe,0xa0,0x82,0x08,0x5d,0xc3,0x55,0x89,
0xe5,0xc6,0x06,0x87,0x08,0x2b,0xc7,0x06,0x8c,0x08,0x00,0x00,0xc7,0x06,0x88,0x08,
0x00,0x00,0xb0,0xff,0xd0,0xd8,0x72,0x03,0xe9,0xa9,0x00,0xff,0x36,0x80,0x08,0xe8,
0x0b,0xff,0xd0,0xd8,0x73,0x1c,0xa1,0x88,0x08,0xb1,0x04,0xd3,0xe0,0x50,0xff,0x36,
0x80,0x08,0xe8,0x3d,0xff,0xb4,0x00,0x59,0x01,0xc8,0xa3,0x88,0x08,0xe8,0xd0,0xfe,
0xeb,0xd9,0x80,0x3e,0x87,0x08,0x2b,0x75,0x09,0xa1,0x88,0x08,0x01,0x06,0x8c,0x08,
0xeb,0x07,0xa1,0x88,0x08,0x29,0x06,0x8c,0x08,0xa0,0x80,0x08,0x3c,0x0d,0xb0,0xff,
0x74,0x01,0x40,0x8a,0x0e,0x80,0x08,0x50,0x80,0xf9,0x3a,0xb0,0xff,0x74,0x01,0x40,
0x5a,0x08,0xd0,0x50,0x80,0xf9,0x2c,0xb0,0xff,0x74,0x01,0x40,0x5a,0x08,0xd0,0x50,

0x80,0xf9,0x20,0xb0,0xff,0x74,0x01,0x40,0x59,0x08,0xc8,0xd0,0xd8,0x73,0x05,0xa1,
0x8c,0x08,0x5d,0xc3,0x80,0x3e,0x80,0x08,0x2b,0xb0,0xff,0x74,0x01,0x40,0x50,0x80,
0x3e,0x80,0x08,0x2d,0xb0,0xff,0x74,0x01,0x40,0x59,0x08,0xc8,0xd0,0xd8,0x72,0x03,
0xe9,0x6a,0xfd,0xa0,0x80,0x08,0xa2,0x87,0x08,0xc6,0x06,0x88,0x08,0x00,0xe8,0x4f,
0xfe,0xe9,0x4e,0xff,0x5d,0xc3,0x55,0x89,0xe5,0x8b,0x46,0x04,0xa3,0x90,0x08,0xe8,
0x2c,0xff,0xa3,0x8e,0x08,0x80,0x3e,0x80,0x08,0x3a,0x75,0x08,0xe8,0x31,0xfe,0xa1,
0x8e,0x08,0xeb,0xe8,0x5d,0xc2,0x02,0x00,0x55,0x89,0xe5,0xc6,0x06,0x83,0x08,0x01,
0xa0,0x83,0x08,0x3a,0x46,0x04,0x77,0x0d,0xb0,0x20,0x50,0xe8,0x90,0xfd,0x80,0x06,
0x83,0x08,0x01,0x75,0xeb,0x5d,0xc2,0x02,0x00,0x55,0x89,0xe5,0xb0,0x0d,0x50,0xe8,
0x7c,0xfd,0xb0,0x0a,0x50,0xe8,0x76,0xfd,0x5d,0xc3,0x55,0x89,0xe5,0xe8,0xe9,0xff,
0xb0,0x02,0x50,0xe8,0xc2,0xff,0x5d,0xc3,0x55,0x89,0xe5,0xe8,0xe2,0xfd,0x80,0x3e,
0x80,0x08,0x20,0x74,0xf6,0xb8,0x00,0x00,0x50,0xe8,0x8a,0xff,0x80,0x3e,0x80,0x08,
0x0d,0x74,0x03,0xe9,0xd7,0xfc,0xe8,0xc0,0xff,0xff,0x16,0x8e,0x08,0x5d,0xc3,0x55,
0x89,0xe5,0xe8,0xbb,0xfd,0x80,0x3e,0x80,0x08,0x20,0x74,0xf6,0xb8,0x00,0x00,0x50,
0xe8,0x63,0xff,0x80,0x3e,0x80,0x08,0x0d,0x74,0x03,0xe9,0xb0,0xfc,0xb0,0xff,0xd0,
0xd8,0x72,0x03,0xe9,0x88,0x00,0xe8,0xa1,0xff,0xff,0x36,0x8e,0x08,0xe8,0x4b,0xfd,
0xb0,0x2d,0x50,0xe8,0x08,0xfd,0xc4,0x1e,0x8e,0x08,0x26,0xff,0x37,0xe8,0x10,0xfd,
0xb0,0x2d,0x50,0xe8,0xf8,0xfc,0xe8,0x77,0xfd,0xa0,0x80,0x08,0x3c,0x2c,0xb0,0xff,
0x75,0x01,0x40,0x50,0x80,0x3e,0x80,0x08,0x20,0xb0,0xff,0x75,0x01,0x40,0x59,0x20,
0xc8,0x50,0x80,0x3e,0x80,0x08,0x0d,0xb0,0xff,0x75,0x01,0x40,0x59,0x20,0xc8,0xd0,
0xd8,0x73,0x33,0xe8,0xc9,0xfd,0xa2,0x8a,0x08,0xa0,0x8a,0x08,0xc4,0x1e,0x8e,0x08,
0x26,0x88,0x07,0xc4,0x1e,0x8e,0x08,0x26,0x8a,0x07,0x3a,0x06,0x8a,0x08,0x74,0x16,
0xb0,0x02,0x50,0xe8,0x02,0xff,0x2e,0x8d,0x06,0x22,0x00,0x0e,0x50,0xb0,0x11,0x50,
0xe8,0xef,0xfc,0xe9,0x27,0xfc,0x83,0x06,0x8e,0x08,0x01,0xe9,0x6f,0xff,0x5d,0xc3,
0x55,0x89,0xe5,0xe8,0x0a,0xfd,0x80,0x3e,0x80,0x08,0x20,0x74,0xf6,0xb8,0x00,0x00,
0x50,0xe8,0xb2,0xfe,0x80,0x3e,0x80,0x08,0x0d,0x75,0x08,0xc7,0x06,0x84,0x08,0x01,
0x00,0xeb,0x1d,0x80,0x3e,0x80,0x08,0x3a,0x75,0x03,0xe9,0xf0,0xfb,0xe8,0xe0,0xfc,
0xe8,0xcb,0xfd,0xa3,0x84,0x08,0x80,0x3e,0x80,0x08,0x0d,0x74,0x03,0xe9,0xdd,0xfb,
0xe8,0xd7,0xfe,0xa1,0x8e,0x08,0x50,0xe8,0x81,0xfc,0xb0,0x03,0x50,0xe8,0x98,0xfe,
0x83,0x3e,0x84,0x08,0x01,0x76,0x0f,0xa1,0x8e,0x08,0x83,0xe0,0x0f,0xb9,0x03,0x00,
0xf7,0xe1,0x50,0xe8,0x82,0xfe,0x83,0x3e,0x84,0x08,0x00,0x76,0x4b,0xc4,0x1e,0x8e,
0x08,0x26,0xff,0x37,0xe8,0x29,0xfc,0xb0,0x20,0x50,0xe8,0x11,0xfc,0x83,0x06,0x8e,
0x08,0x01,0xa1,0x84,0x08,0x83,0xe8,0x01,0xa3,0x84,0x08,0x09,0xc0,0xb0,0xff,0x77,
0x01,0x40,0x8b,0x0e,0x8e,0x08,0x83,0xe1,0x0f,0x50,0x09,0xc9,0xb0,0xff,0x74,0x01,
0x40,0x59,0x20,0xc8,0xd0,0xd8,0x73,0xbe,0xe8,0x6f,0xfe,0xff,0x36,0x8e,0x08,0xe8,
0x19,0xfc,0xb0,0x03,0xb4,0x00,0xeb,0xaa,0x5d,0xc3,0x55,0x89,0xe5,0xe8,0x49,0xfe,
0xb0,0x2e,0x50,0xe8,0xc8,0xfb,0xe8,0x47,0xfc,0xc6,0x06,0x83,0x08,0x00,0x80,0x3e,
0x83,0x08,0x02,0x76,0x03,0xe9,0x45,0xfb,0xa0,0x80,0x08,0x8a,0x1e,0x83,0x08,0xb7,
0x00,0x2e,0x3a,0x87,0x10,0x00,0x75,0x05,0xa0,0x83,0x08,0x5d,0xc3,0x80,0x06,0x83,
0x08,0x01,0x75,0xda,0xe9,0x26,0xfb,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x06,0x1e,0x2e,0x8e,0x1e,0x37,0x00,0x50,0x51,0x52,0x53,0x56,0x57,0xe8,0x09,0x00,
0x5f,0x5e,0x5b,0x5a,0x59,0x58,0x1f,0x07,0xcf,0x55,0x89,0xe5,0x2e,0x8d,0x06,0xf6,
0x07,0x0e,0x50,0xb0,0x0c,0x50,0xe8,0xb9,0xfb,0xe9,0xb9,0xfa,0x06,0x1e,0x2e,0x8e,
0x1e,0x37,0x00,0x50,0x51,0x52,0x53,0x56,0x57,0xe8,0x09,0x00,0x5f,0x5e,0x5b,0x5a,

0x59,0x58,0x1f,0x07,0xcf,0x55,0x89,0xe5,0x2e,0x8d,0x06,0x02,0x08,0x0e,0x50,0xb0,
0x11,0x50,0xe8,0x8d,0xfb,0xe8,0xc2,0xfd,0x5d,0xc3,0x06,0x1e,0x2e,0x8e,0x1e,0x37,
0x00,0x50,0x51,0x52,0x53,0x56,0x57,0xe8,0x09,0x00,0x5f,0x5e,0x5b,0x5a,0x59,0x58,
0x1f,0x07,0xcf,0x55,0x89,0xe5,0x2e,0x8d,0x06,0x13,0x08,0x0e,0x50,0xb0,0x08,0x50,
0xe8,0x5f,0xfb,0xe9,0x5f,0xfa,0x06,0x1e,0x2e,0x8e,0x1e,0x37,0x00,0x50,0x51,0x52,
0x53,0x56,0x57,0xe8,0x09,0x00,0x5f,0x5e,0x5b,0x5a,0x59,0x58,0x1f,0x07,0xcf,0x55,
0x89,0xe5,0x2e,0x8d,0x06,0x1b,0x08,0x0e,0x50,0xb0,0x0c,0x50,0xe8,0x33,0xfb,0xe8,
0x68,0xfd,0x5d,0xc3,0x06,0x1e,0x2e,0x8e,0x1e,0x37,0x00,0x50,0x51,0x52,0x53,0x56,
0x57,0xe8,0x09,0x00,0x5f,0x5e,0x5b,0x5a,0x59,0x58,0x1f,0x07,0xcf,0x55,0x89,0xe5,
0x2e,0x8d,0x06,0x27,0x08,0x0e,0x50,0xb0,0x0c,0x50,0xe8,0x05,0xfb,0xe8,0x3a,0xfd,
0x5d,0xc3,0x06,0x1e,0x2e,0x8e,0x1e,0x37,0x00,0x50,0x51,0x52,0x53,0x56,0x57,0xe8,
0x09,0x00,0x5f,0x5e,0x5b,0x5a,0x59,0x58,0x1f,0x07,0xcf,0x55,0x89,0xe5,0x2e,0x8d,
0x06,0x33,0x08,0x0e,0x50,0xb0,0x0c,0x50,0xe8,0xd7,0xfa,0xe8,0x0c,0xfd,0x5d,0xc3,
0x06,0x1e,0x2e,0x8e,0x1e,0x37,0x00,0x50,0x51,0x52,0x53,0x56,0x57,0xe8,0x09,0x00,
0x5f,0x5e,0x5b,0x5a,0x59,0x58,0x1f,0x07,0xcf,0x55,0x89,0xe5,0x2e,0x8d,0x06,0x3f,
0x08,0x0e,0x50,0xb0,0x0c,0x50,0xe8,0xa9,0xfa,0xe8,0xde,0xfc,0x5d,0xc3,0x06,0x1e,
0x2e,0x8e,0x1e,0x37,0x00,0x50,0x51,0x52,0x53,0x56,0x57,0xe8,0x09,0x00,0x5f,0x5e,
0x5b,0x5a,0x59,0x58,0x1f,0x07,0xcf,0x55,0x89,0xe5,0x2e,0x8d,0x06,0x4b,0x08,0x0e,
0x50,0xb0,0x0c,0x50,0xe8,0x7b,0xfa,0xe8,0xb0,0xfc,0x5d,0xc3,0x06,0x1e,0x2e,0x8e,
0x1e,0x37,0x00,0x50,0x51,0x52,0x53,0x56,0x57,0xe8,0x09,0x00,0x5f,0x5e,0x5b,0x5a,
0x59,0x58,0x1f,0x07,0xcf,0x55,0x89,0xe5,0x2e,0x8d,0x06,0x57,0x08,0x0e,0x50,0xb0,
0x0c,0x50,0xe8,0x4d,0xfa,0xe8,0x82,0xfc,0x5d,0xc3,0x06,0x1e,0x2e,0x8e,0x1e,0x37,
0x00,0x50,0x51,0x52,0x53,0x56,0x57,0xe8,0x09,0x00,0x5f,0x5e,0x5b,0x5a,0x59,0x58,
0x1f,0x07,0xcf,0x55,0x89,0xe5,0x2e,0x8d,0x06,0x63,0x08,0x0e,0x50,0xb0,0x0c,0x50,
0xe8,0x1f,0xfa,0xe8,0x54,0xfc,0x5d,0xc3,0x06,0x1e,0x2e,0x8e,0x1e,0x37,0x00,0x50,
0x51,0x52,0x53,0x56,0x57,0xe8,0x09,0x00,0x5f,0x5e,0x5b,0x5a,0x59,0x58,0x1f,0x07,
0xcf,0x55,0x89,0xe5,0x2e,0x8d,0x06,0x6f,0x08,0x0e,0x50,0xb0,0x0c,0x50,0xe8,0xf1,
0xf9,0xe8,0x26,0xfc,0x5d,0xc3,0x44,0x49,0x56,0x49,0x44,0x45,0x20,0x45,0x52,0x52,
0x4f,0x52,0x4e,0x4f,0x4e,0x20,0x4d,0x41,0x53,0x4b,0x41,0x42,0x4c,0x45,0x20,0x49,
0x4e,0x54,0x52,0x4f,0x56,0x45,0x52,0x46,0x4c,0x4f,0x57,0x49,0x4e,0x54,0x45,0x52,
0x52,0x55,0x50,0x54,0x20,0x33,0x32,0x49,0x4e,0x54,0x45,0x52,0x52,0x55,0x50,0x54,
0x20,0x33,0x33,0x49,0x4e,0x54,0x45,0x52,0x52,0x55,0x50,0x54,0x20,0x33,0x34,0x49,
0x4e,0x54,0x45,0x52,0x52,0x55,0x50,0x54,0x20,0x33,0x35,0x49,0x4e,0x54,0x45,0x52,
0x52,0x55,0x50,0x54,0x20,0x33,0x36,0x49,0x4e,0x54,0x45,0x52,0x52,0x55,0x50,0x54,
0x20,0x33,0x37,0x49,0x4e,0x54,0x45,0x52,0x52,0x55,0x50,0x54,0x20,0x33,0x38,0x49,
0x4e,0x54,0x45,0x52,0x52,0x55,0x50,0x54,0x20,0x33,0x39,0x00,0x00,0x00,0x00,0x00};/
/0x870 - 0x87f

Appendix D. Octave (MATLAB) Utility Scripts

D.1 Utility script to read NASM binary code and convert to hex for EEPROM programmer

```
# MakeInclude1.m
# John Smigel

clear all
close all

fidin = fopen ('8086TestUart5');
bincode = fread (fidin);

# convert binary code to hex format
nbytes = length(bincode);
stringcode =
sprintf("0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x
%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,\n",bincode)
fclose(fidin);


 # convert IVT from word to byte

 IVT =
uint16([0x600,0xff60,0x686,0xff60,0x62c,0xff60,0x686,0xff60,0x65A,0xff60,0x686,0xff60,0x686
,0xff60,0x686,0xff60,0x686,0xff60,0x6b4,0xff60,0x6e2,0xff60,0x710,0xff60,0x73e,0xff60,0x76c,
0xff60,0x79a,0xff60,0x7c8,0xff60]);

 IVTB = dec2bin(IVT);

 inindx = 1;
 for idx = 1:2:64
   IVT8(1,idx) = bitand(IVT(1,inindx),0x00ff);
   IVT8(1,idx+1) = bitand(IVT(1,inindx),0xff00);
   inindx = inindx + 1;
 endfor

 for idx = 2:2:64
   IVT8(1,idx) = bitshift(IVT8(1,idx),-8);
 endfor
 stringcode = sprintf("0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,0x%.2x,\n",IVT8)
```

D.2 Utility Script to Convert Logic Analyzer Captures of Data, Address and Control to Readable Format

```
# ViewBinary16C
# version with control inputs
# John Smigel
addpath ../../Documents/LaCaptures
clear all
close all

#Address
fidinA = fopen('2022-02-20_A0to15-2.bin');

bincodeA = fread(fidinA);
fclose(fidinA);

#Data
fidinD = fopen('2022-02-20_D0to15-2.bin');
bincodeD = fread(fidinD);

# convert binary code to hex format
fclose(fidinD);

fidinC = fopen('2022-02-19_CD8to15-1.bin');
bincodeC = fread(fidinC);
fclose(fidinC);


Aoff=0; #needs to be even offset
Doff=0;
Coff=0;

for i=1:2:1000
  # the upper 8 bits of control and data is high data
  if (i+1+Coff) < 1
    highdata2 = 0;
    control = 0;
    else
      highdata2 = bincodeC(i+1+Coff,1);
      control = bincodeC(i+Coff,1);
  endif

# bit 15 is ~BHE, so want to sign extend from A14
  high12 = bincodeA(i+1+Aoff,1);
```

```
  # decode control bits
  #0=~ALE,1=M/~IO,2=~WR,3=~RD,4=~SROM,5=DISABLE,6=~SEL550, 7=~SRAM
  notALE = bitand(control,0x1);
  notmio = bitand(control,0x2)/2;
  notwr = bitand(control,0x4)/4;
  notrd = bitand(control,0x8)/8;
  notsrom = bitand(control,0x10)/16;
  disable = bitand(control,0x20)/32;
  notsel550 = bitand(control,0x40)/64;
  notsram = bitand(control,0x80)/128;
  #mask off ~BHE and sign extend A15=A14
  notbhe = bitand(high12,0x80)/128;
  a14 = bitand(high12,0x40);
  high12 = bitand(high12,0x7f);
  high12 = bitor(high12,a14*2);
  printf("%4d %.2x %.1x %.1x %.1x %.1x   %.1x %.1x %.1x %.1x  %.1x   %.2x%.2x  %.2x  %.2x%.2x
%.2x%.2x\n",

i,control,notsram,notsel550,disable,notsrom,notrd,notwr,notmio,notALE,notbhe,high12,bincod
eA(i+Aoff,1),highdata2,
  bincodeD(i+1+Doff,1),bincodeD(i+Doff,1), bincodeD(i+Doff),bincodeD(i+1+Doff))
  endfor
```