iSBC 80/10A — SYSTEM 80/10 Single Board Computer Applications

Thomas Rolander
Microcomputer Applications

iSBC 80/10A-SYSTEM 80/10 Single Board Computer Applications

Contents

INTRODUCTION

The recent entry of the single board computer into the broad field of electronic applications is substantiating the billing as a "super component". Single board computers provide a solution to several problems that have not been solved by the use of conventional computers: cost, size, and design specialization.

Many potential microcomputer applications have been overlooked because of the design tasks required to build a microcomputer system. These tasks traditionally include interfacing of the system clock, read/write memory, I/O ports and drivers, serial communications interface, bus control logic and drivers. Intel's iSBC 80/10A enables the design engineer to concentrate on the application of microcomputers, rather than on implementation details.

This application note begins with an overview of the Intel® iSBC 80/10A. Readers who are familiar with the iSBC 80/10A may choose to skip to the applications section, which describes the following typical iSBC 80/10A applications:

- The iSBC 80/10A used for instrumentation control of a Fluke 8375 Digital Multimeter.
- The iSBC 80/10A used as a SCADA Terminal in a communication application.
- The iSBC 80/10A used for temperature monitoring in a process control application.
- The iSBC 80/10A used as an interrupt driven device controller for a Centronics printer.

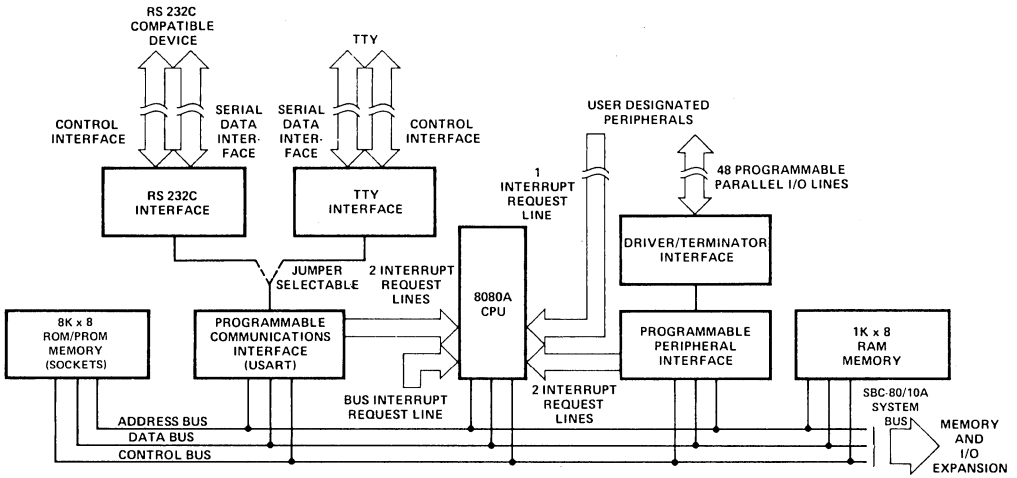
Each example shows the user program and hardware required for the application. The program listings are interspersed with the text describing the application. Both 8080 Macro Assembly Language and Intel's PL/M-80 are used in the examples.

The software was developed on an Intel® Microcomputer Development System (MDS). The MDS provided the tools necessary to edit, assemble or compile, link and locate the application software. Hardware development was facilitated by the use of Intel's In-Circuit Emulator (ICE 80). For further information regarding the Microcomputer Development System, the reader is referred to the publications listed at the beginning of this application note.

OVERVIEW

The iSBC 80/10A is a member of Intel's complete line of OEM computer systems which take full advantage of Intel's LSI technology to provide economical, self-contained computer based solutions for OEM applications. The iSBC 80/10A is a complete computer system on a single 6.75-by-12 inch printed circuit card. A block diagram of the iSBC 80/10A is shown in Figure 1.

Intel's powerful 8-bit n-channel MOS 8080A CPU, fabricated on a single LSI chip, is the central processor for the iSBC 80/10A. The 8080A contains six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or in pairs, providing both single and double precision operators.



1. Interrupts originating from the Programmable Communications Interface and Programmable Peripheral Interface are jumper selectable.

Figure 1. iSBC 80/10A Block Diagram

The 8080A has a 16-bit program counter which allows direct addressing of up to 64K bytes of memory. An external stack, located within any portion of read/write memory, may be used as a last in/first out stack to store the contents of the program counter, flags, accumulator and all of the six general purpose registers. A 16-bit stack pointer addresses the external stack. This provides subroutine nesting that is bounded only by memory size.

The iSBC 80/10A contains 1K bytes of read/write memory using Intel's low power static RAM. All on board RAM read and write operations are performed at maximum processor speed. Four sockets for up to 8K bytes of non-volatile read-only memory are provided on the board. Read-only memory may be added in 1K byte increments (up to 4K total) using Intel® 8708 erasable and electrically reprogrammable ROMs (EPROMs) or Intel 8308 masked ROMs. Optionally, if more than 4K bytes are required, read only memory may be added in 2K byte increments (up to 8K total) using Intel® 2716 EPROMs or 2316E masked ROMs. All on-board ROM or EPROM read operations are performed at maximum processor speed.

The iSBC 80/10A contains 48 programmable parallel I/O lines implemented using two Intel® 8255 Programmable Peripheral Interfaces. The system software is used to configure the I/O lines in any combination of unidirectional input/output, and bidirectional ports indicated in Table I. Therefore, the I/O interface may be customized to meet specific peripheral requirements. To support the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the I/O interface provides the appropriate combination of optional line drivers and terminators to allow the required sink current, polarity, and drive/termination characteristics for each application. The 48 programmable I/O lines and signal ground lines are brought out to two 50-pin edge connectors that mate with flat, round, or woven cable.

A programmable communications interface using Intel's 8251 Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is contained on the iSBC 80/10A. A jumper selectable baud rate generator provides the 8251 with all common communication frequencies. The 8251 can be programmed by the user's system software to select the desired asynchronous or synchronous serial data transmission technique (including IBM Bi-sync). The mode of operation (synchronous or asynchronous), data format, control character format, parity, and asynchronous transmission rate are all under program control. The 8251 provides full duplex, double buffered transmission and receive capability. Parity, overrun, and framing error detection circuits are all incorporated in the 8251. The inclusion of jumper selectable TTY or EIA RS232C compatible interfaces on the board, in conjunction with the 8251, provide a direct interface to teletypes, CRTs, asynchronous and synchronous modems, and other RS232C compatible devices. The RS232C or TTY command lines, serial data lines, and signal ground lines are brought out to a 25-pin edge connector that mates with RS232C compatible flat, round, or woven cable.

Interrupt requests may originate from six sources. Two from the 8255's, two from the 8251 and two from user designated peripheral devices.

TABLE 1 INPUT/OUTPUT PORT MODES OF OPERATION

PORT	NO. OF LINES	MODE OF OPERATION					
		UNIDIRECTIONAL				BIDIRECTIONAL	CONTROL
		INPUT		OUTPUT			
		UNLATCHED	LATCHED & STROBED	LATCHED	LATCHED & STROBED		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	8	X		X			X[1]
4	8	X		X			
5	8	X		X			
6	4	X		X			
	4	X		X			

1. Note: Port 3 must be used as a control port when either Port 1 or Port 2 are used as a latched and strobed input or a latched and strobed output or Port 1 is used as a bidirectional port.

The 8255's can generate interrupts when a byte of information is ready to be transferred to the CPU (i.e., input buffer full) or a byte of information has been transferred to a peripheral device (i.e., output buffer is empty).

The 8251 can generate interrupts when a character is ready to be transferred to the CPU (i.e., receive channel buffer is full) or a character is ready to be transmitted (i.e., transmit channel data buffer is empty).

The user designated peripheral devices can generate two interrupts: one via the system bus and the other via the I/O edge connector.

The two interrupts from the 8255's and the two interrupts from the 8251 are all individually maskable under program control. The six interrupt request lines share a single CPU interrupt level. When an interrupt request is recognized, a RESTART 7 instruction is generated. The processor responds by suspending program execution and making a subroutine call to a user defined interrupt service routine originating at location 38 (Hexadecimal).

iSBC 80/10A memory and I/O capacity may be increased by adding standard Intel memory and I/O boards. Modular expandable backplanes and card cages, each with a four-board capacity, are available to support multi-board systems.

The development cycle of iSBC 80/10A based products may be significantly reduced using the Intellec Microcomputer Development System. The resident macro-assembler, PL/M-80 compiler, text editor, and system monitor greatly simplify the design, development, and debug of user designed iSBC 80/10A system software. A diskette-based system allows programs to be loaded, assembled, edited, and executed faster than using conventional paper tape, card, or cassette peripherals. A unique In-Circuit Emulator (ICE 80) provides the capability of developing and debugging software directly on the iSBC 80/10A.

iSBC CONFIGURATION OPTIONS

The iSBC 80/10 provides the user with a powerful and flexible I/O capability for both parallel and serial transfers. This section discusses the user programmable and jumper-selectable options, and bus interfacing.

SERIAL I/O OPTIONS

The serial I/O interface, using Intel's 8251 USART, provides a serial data communications channel that can be programmed to operate with most of the current serial data transmission protocols. There are three general areas of serial I/O options:

1. choice of interface type, RS232C or current loop,
2. baud rate and program-selectable mode options,
3. choice of an interrupt mechanism.

The user has the choice, through jumper connections, of configuring the serial I/O logic to present either an RS232C or a 20 mA current loop interface to an external device. If an RS232C interface is used, the 8251 can assume the role of a "data set" or a "data processing terminal". This enables the serial interface to be connected to different devices such as modems and computer terminals.

There are two factors which enter into the choice of baud rate. They are the actual clock frequency used to drive the transmit/receive clocks on the 8251 and the baud rate factor selected by a programmable mode instruction control word output by the processor to the 8251. The baud rate factor is used to effectively divide the 8251 transmit and receive clocks by 1, 16 or 64. During normal operation a factor of 16 is selected for asynchronous transmissions from 9.6K to 300 baud. A factor of 64 must be used to achieve a baud rate of 110. The baud rate factor is only applicable to asynchronous transmission, as all synchronous transmission is done with an implied factor of one.

Before beginning serial I/O operations, the 8251 must be program-initialized to support the desired mode of operation. The CPU initializes the 8251 by issuing a set of control bytes to the USART device. These control words specify:

- synchronous or asynchronous operation
- baud rate factor
- character length
- number of stop bits
- even/odd parity
- parity/no parity

Refer to the *iSBC 80/10 and iSBC 80/10A Single Board Computer Hardware Reference Manual* or the "8251 Application Note" for details on the control words used to direct the operation of the 8251.

The serial I/O logic can be configured with different forms of interrupt request mechanisms. By connecting a jumper, the user can allow the 8251's Receiver Ready output to generate an interrupt request. The Receiver Ready output goes high whenever the Receiver Enable bit of the command

word has been set and the 8251 contains a character that is ready to be input to the CPU. The user can also choose to have the 8251's Transmitter Ready or Transmitter Empty output activate the interrupt request. The Transmitter Empty goes high when the 8251 has no characters to transmit. Transmitter Ready is high when the 8251 is ready to accept a character from the CPU. Both Transmitter Empty and Transmitter Ready are enabled by setting the Transmit Enable bit of the command word. Upon receiving an interrupt, the program can determine the actual condition which is responsible for the interrupt by reading the status of the 8251 device.

PARALLEL I/O OPTIONS

The parallel I/O interface consists of six 8-bit I/O ports implemented with two Intel 8255 Programmable Peripheral Interface devices. Eight lines already have a bidirectional driver and termination network permanently installed. The remaining 40 lines are uncommitted. Sockets are provided for the installation of active driver networks or passive termination networks as required to meet the specific needs of the user system.

The primary considerations in determining how to use each of the six I/O ports are:

1. choice of operating mode,
2. direction of data flow (input, output or bidirectional),
3. selection of interrupt mechanism,
4. choice of driver/termination networks for the port's data path.

Operating Modes. There are three basic operating modes that can be selected by the system software. The modes of operation will be described here in general terms, leaving it to the reader to obtain details from the *iSBC 80/10 and iSBC 80/10A Single Board Computer Hardware Reference Manual* or the "8255 Application Note."

Mode 0 is a basic input/output functional configuration which provides simple input and output operations. No "handshaking" is required, data is simply written to or read from a specified port. The outputs are latched and the inputs are unlatched.

Mode 1 is a strobed input/output functional configuration which provides a means for transferring I/O data to or from a specified port in conjunction with strobes or handshaking signals. The outputs are latched and are accompanied by an output control line which indicates that the processor has loaded the output port with a data byte. The input data is latched when accompanied by its externally operated control signal.

Mode 2 is a strobed bidirectional bus input/output functional configuration which provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data. Handshaking signals are provided to maintain proper bus flow discipline in a manner similar to mode 1.

Data Flow Direction. In addition to the choice of operating mode, the user may also specify the direction of data flow, input or output from the 8255's. At the time of RESET, the 8255's are configured into the input mode until altered by a control word directed to the control word register. When an output mode control word is received, all of the data bits are set to the low output state.

Interrupt Mechanism. When the 8255 is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from one of the ports (port C), can be inhibited or enabled by setting or resetting the associated interrupt enable flip-flop, using the bit set/reset function of port C. This function allows the programmer to mask the interrupts from specific I/O devices without affecting any other device in the interrupt structure.

Driver/Termination Networks. Depending on the direction of data flow, the user will select the appropriate TTL line drivers and Intel terminators that are compatible with the I/O driver/terminator sockets on the iSBC 80/10A. The list of suitable line drivers includes those with inverting, noninverting, and open collector characteristics. There are two types of terminators: a 220-ohm/330-ohm divider or a 1K ohm pull-up.

BUS INTERFACING

The system bus interface logic consists of three general groups of circuitry:

1. gates that accept the various bus control signals, the interrupt request lines, and the ready indications, and then apply these signals to the CPU logic elements,
2. the system bus drivers,
3. the failsafe circuitry which generates an acknowledgment during interrupt sequences and during those cycles in which an ac-

knowledgment is not returned because a non-existent device was inadvertently addressed.

Bus Interface Signals. The following paragraphs describe portions of the system bus interfacing logic relevant to interfacing a user device to the iSBC 80/10A. (Note: Whenever a signal is active-low, its mnemonic is followed by a slash; for example, MRDC/ means that the level on that line will be low when the memory read command is true.)

BCLK/ – Bus clock; used to synchronize bus control circuits on all master modules. BCLK/ has a frequency of 9.216 MHz. BCLK/ may be slowed, stopped or single stepped, if desired.

INIT/ – Initialization signal; resets the entire system to a known internal state.

BPRN – Bus priority input signal; indicates to the iSBC 80/10A that a higher priority master module is requesting use of the system bus. BPRN suspends the processing activity and drivers of the iSBC 80/10A until the signal goes low.

BUSY/ – Bus busy signal; indicates that the bus is currently in use. BUSY/ prevents all other master modules from gaining control of the bus. BUSY/ is driven by the HLDA/ output from the iSBC 80/10A in response to a BPRN input. It indicates that the bus is available.

MRDC/ – Memory read command; indicates that the address of a memory location has been placed on the system address lines and specifies that the contents of the addressed location are to be read and placed on the system data bus.

MWTC/ – Memory write command; indicates that the address of a memory location has been placed on the system address lines and that a data word has been placed on the system data bus. MWTC/ specifies that the data word is to be written into the addressed memory location.

IORC/ – I/O read command; indicates that the address of an input port has been placed on the system address bus and that the data at that input port is to be read and placed on the system data bus.

IOWC/ – I/O write command; indicates that the address of an output port has been placed on the system address bus and that the contents of the system data bus are to be output to the addressed port.

XACK/ – Transfer acknowledge signal; the required response of an external memory location or I/O port which indicates that the specified read/write operation has been completed (that is, data has been placed on, or accepted from, the system data bus lines).

AACK/ – An advance acknowledge, in response to a memory read or write command, that allows the memory to complete the specified operation without requiring the CPU to wait.

CCLK/ – Constant clock; provides a clock signal of constant frequency (9.216 MHz) for use by optional memory and I/O expansion boards. The same signal is used to drive both CCLK/ and BCLK/.

INTR1/ – Externally generated interrupt request.

ADR0/–ADRF/ – 16 Address lines; used to transmit the address of the memory location or I/O port to be accessed. ADRF/ is the most significant bit.

DAT0/–DAT7/ – Bidirectional data lines; used to transmit/receive information to/from a memory location or I/O port. DAT7/ is the most significant bit.

Bus Acknowledges. Further distinction between transfer acknowledge (XACK/) and advance acknowledge (AACK/) is required. All external memory and I/O transfer requests must return XACK/ to the iSBC 80/10A (even if AACK/ is also returned). XACK/ indicates that data has been placed on (read command) or accepted from (write command) the system data bus lines. AACK/ is an advance acknowledge in response to a memory or I/O port command. It has been provided because the 8080A samples the ready line before valid data is required on the bus. If this condition is properly anticipated, AACK/ can be returned before the data is actually read, thus allowing an earlier operation to be completed. AACK/ should be used only with a thorough understanding of the additional information provided in the *iSBC 80/10 and iSBC 80/10A Single Board Computer Hardware Reference Manual. DMA Transfers.* An external device can make DMA transfers to or from RAM expansion boards. The transfer is coordinated with the iSBC 80/10A by means of two bus signals: bus priority input (BPRN) and bus busy (BUSY/). The first step in making a DMA transfer is to obtain control of the system bus. This is

achieved by asserting BRPN to the iSBC 80/10A and then waiting until the iSBC 80/10A returns BUSY/, indicating that it has relinquished control of the system bus. When this step is completed the external device may proceed with its DMA transfers until it is finished. At that time BPRN should be removed to allow the iSBC 80/10A to regain control of the system bus. It should be noted that the iSBC 80/10A is placed in a hold state when it does not have control of the system bus.

APPLICATIONS

The iSBC 80/10A may be applied to a wide variety of applications. Specific applications in four areas are presented in this application note. They are presented to illustrate a broad spectrum of single board computer capabilities and to demonstrate the use of various system features.

INSTRUMENTATION

Microprocessors have been used in instrumentation for many tasks ranging from handling simple interface functions to control of the analog to digital conversion process. The use of a single board computer can further serve in the application of instruments themselves to laboratory or process control environments. It is quite often necessary in these applications to control instrumentation remotely. A number of rather expensive minicomputer-controlled solutions now exist on the market as automatic test equipment (ATE) systems. The iSBC 80/10A presents itself as a cost effective solution in situations where the larger ATE systems are beyond economic justification.

The iSBC 80/10A can be the sole CPU element in the system, providing instrumentation control and computational capability; or it can supplement a larger host CPU by handling distributed processing requirements.

Instrumentation Control Application Example

Most instruments such as DVMs, counters, data loggers, synthesizers, etc., have optional data output units (DOUs) and/or remote control units (RCUs). It is particularly time consuming to interface each instrument's DOU/RCU with custom-digital logic. Until the recent IEEE-488 interface standard, there was little in common from one interface to the next. The parallel I/O lines of the iSBC 80/10A provide a common interface element that can be adapted to a majority of the DOUs and RCUs available today by means of software.

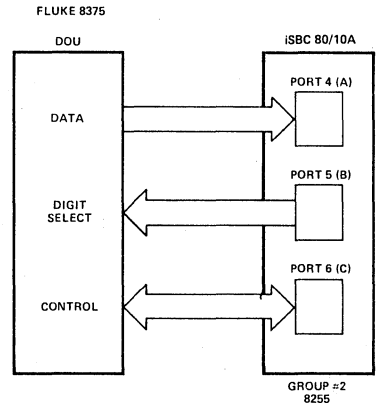


Figure 2. Interface Block Diagram

This instrumentation control application shows how the iSBC 80/10A has been used to control and read the data from the data output unit (DOU) of a Fluke 8375 Digital Multimeter.

Interfacing the iSBC 80/10A to the Fluke 8375 DOU has been accomplished through the use of three parallel I/O ports shown in Figure 2. An 8-bit port has been used to read input data from the Fluke 8375 DOU. Another 8-bit port has been used to control the multiplexing of data from the DOU to the iSBC 80/10A. And, an 8-bit port has been used to provide the required control and monitoring of the following DOU functions: busy flag, sample sync flag, timeout enable, external trigger and trigger inhibit.

The following listing contains a complete program to provide the necessary interface control functions as well as an exercise program. The program listing is interspersed with text that is used to clarify the elements of the program.

```
0 ;
1 ;   INSTRUMENTATION CONTROL APPLICATION
2 ;
3 ;      FLUKE 8375 DIGITAL MULTIMETER
4 ;
5 ;   DATA OUTPUT UNIT (DOU) CONTROLLER
6 ;
7 ;
8
```

The CSEG directs the ISIS-II 8080 Assembler to generate a relocatable code segment. Relocatable code can later be placed at any memory address by Intel's LOCATE program. This lets you write your program without worrying about the application's final memory configuration.

```
9
10 ;
11        CSEG
12 ;
13
```

Equate Table. The following table is used to give symbolic names to the binary I/O port addresses. The names used later in the program increase readability.

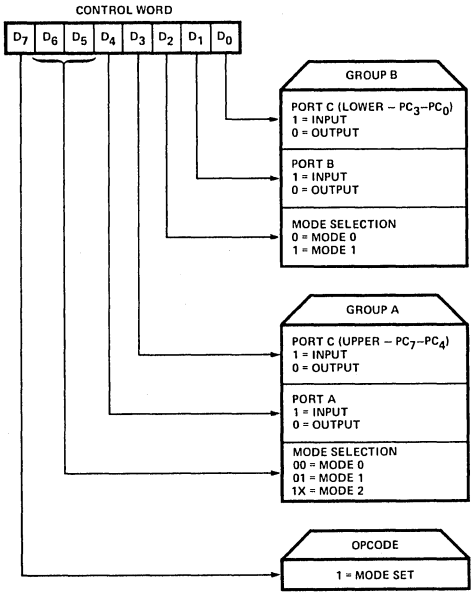
```
14
15 ;
16 ;   EQUATE TABLE
17 ;
18 CWR     EQU     0EBH    ; 8255 #2 CONTROL WORD REGISTER
19 DATIN   EQU     0E8H    ; DECADE PAIR DATA INPUT PORT
20 STB     EQU     0E9H    ; STROBE OUTPUT PORT
21 FLG     EQU     0EAH    ; FLAG INPUT PORT
22 TRG     EQU     0EAH    ; TRIGGER OUTPUT PORT
23 ;
24
```

The exercise program uses some of the subroutines provided in the iSBC 80/10A System Monitor PROMs. The addresses of the subroutines are included in the equate table.

```
25
26 ;
27 GETCH   EQU     0220H   ; GET CONSOLE INPUT, MASK OFF PARITY
28 CO      EQU     01E8H   ; CONSOLE OUTPUT
29 CROUT   EQU     01F3H   ; PRINT <CR><LF>
30 NMOUT   EQU     02C2H   ; DISPLAY BYTE IN ACCUM
31 ;
32
```

The use of the iSBC 80/10A parallel I/O ports requires that the mode of operation be defined for each port. This is typically done by an initialization subroutine executed when the iSBC 80/10A is powered up or reset.

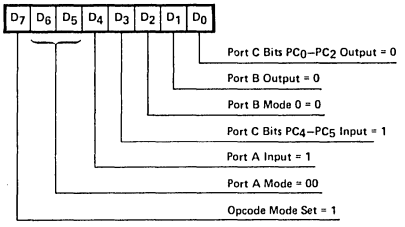
8255 Control Word. When the opcode field (bit 7) of a control word directed to an 8255 is equal to one, the control word is interpreted as a mode definition control word. The mode definition control word format is shown below:



Observing the schematic for the iSBC 80/10A – Fluke 8375 DOU (Figure 3), it can be seen that the 8255 #2 should be configured through the use of the mode control word as:

Port 4 (A)	Mode 0 Input
Port 5 (B)	Mode 0 Output
Port 6 (C)	Bits PC2–PC0 Output
Port 6 (C)	Bits PC5–PC4 Input

The following mode control word is used:



Mode Control Word = 1001 1000 Binary = 98H

```
33
34 ;
35 ; *** 8255 #2 INITIALIZATION SUBROUTINE
36 ;
37 INIT:
38          MVI     A,10011000B     ; LD MODE CONTROL WORD
39          OUT     CWR             ; OUTPUT TO 8255#2 CNTL WD REG
40 ;
41
```

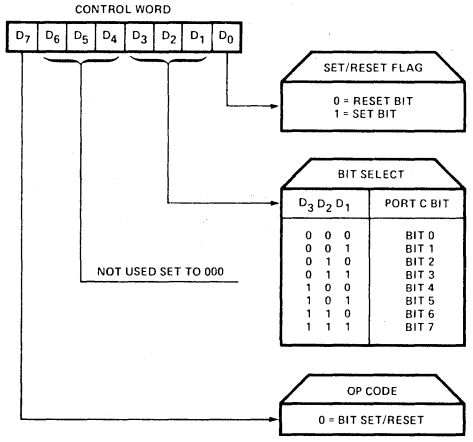
This coding loads the mode control word into the 8255 #2 control word register. Additional initialization code is required to set the strobe and trigger output ports to an inactive state. The schematic shows that inverting drivers have been used for both the strobes and the trigger. When a command is issued to place port 5 (B) into the output mode, bits PB7–PB0 are set to the low output state. Because the low outputs are then inverted and used as strobes to the Fluke 8375, they must then be disabled. The initialization subroutine concludes by disabling the strobes and trigger. The strobes are signals to the DOU which enable its drivers to send data to the iSBC 80/10A. The trigger is a signal to the DOU that the Fluke 8375 should take a reading.

```
42
43 ;
44          MVI     A,0FFH          ; LD MASK TO:
45          OUT     STB             ;   DISABLE STROBES
46          OUT     TRG             ;   DISABLE TRIGGER
47          RET
48 ;
49
```

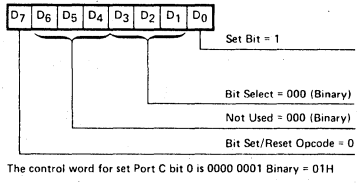
External Trigger Control. Two subroutines are implemented to enable and disable the external trigger mode of the instrument. These subroutines use the bit set/reset capability of the 8255 to independently set or reset three control lines of the Fluke 8375 DOU.

When the opcode field (bit 7) of an 8255 control word equals zero, the control word is a port 6 (C) bit set/reset command word.

The bit set/reset control word format is shown below:



The following example demonstrates how the port 6 (C) bit set/reset control word is constructed to disable the Fluke 8375 external trigger. Note from the schematic (Figure 3) that port 6 (C) bit 0 controls the inhibit external trigger line.



The control word for set Port C bit 0 is 0000 0001 Binary = 01H

```
50
51 ;
52 ; *** ENABLE EXTERNAL TRIGGER SUBROUTINE ***
53 ;
54 ETRIG:
55          MVI     A,00000000B     ; LD RESET BIT 0 CONTROL WORD
56          OUT     CWR             ; OUTPUT TO 8255#2 CNTL WD REG
57          RET
58 ;
59 ; *** DISABLE EXTERNAL TRIGGER SUBROUTINE ***
60 ;
61 DTRIG:
62          MVI     A,00000001B     ; LD SET BIT 0 CONTROL WORD
63          OUT     CWR             ; OUTPUT TO 8255#2 CNTL WD REG
64          RET
65 ;
66
```

Subroutines to enable and disable the timeouts are written in an analogous fashion. The timeout enable line is controlled by port 6 (C) bit 2.

```
67
68 ;
69 ; *** ENABLE TIMEOUTS SUBROUTINE ***
70 ;
71 EPOS:
72          MVI     A,00000101B     ; LD SET BIT 2 CONTROL WORD
73          OUT     CWR             ; OUTPUT TO 8255#2 CNTL WD REG
74          RET
75 ;
76 ; *** DISABLE TIMEOUTS SUBROUTINE ***
77 ;
78 DPOS:
79          MVI     A,00000100B     ; LD RESET BIT 2 CONTROL WORD
80          OUT     CWR             ; OUTPUT TO 8255#2 CNTL WD REG
81          RET
82 ;
83
```

Obtaining Readings. The Fluke 8375 DOU allows readings to be taken in one of two modes. The first, a triggered mode, assumes that the external triggering has not been inhibited and requires the positive edge of a pulse with a minimum width of 1 microsecond on the trigger input. Setting and resetting the port 6 (C) bit 1 produces the 8375 external trigger. After a reading is triggered the 8375 busy flag is tested until the not busy state is reached. At that time the reading that was triggered can be read by the iSBC 80/10A. The last statement in this routine jumps to TKDATA which reads the data from the DOU and then executes the return.

```
84
85 ;
86 ; *** SUBROUTINE TO TAKE EXTERNALLY TRIGGERED READING ***
87 ;
88 TRGR:
89          MVI     A,00000010B     ; LD RESET BIT 1 CONTROL WORD
90          OUT     CWR             ; OUTPUT TO 8255#2 CNTL WD REG
91          INR     A               ; MODIFY CONTROL WORD TO SET BIT 1
92          OUT     CWR             ; OUTPUT TO 8255#2 CNTL WD REG
93 TWT:
94          IN      FLG             ; INPUT THE BUSY FLAG
95          ANI     00100000B       ; TEST PORT C BIT 5
96          JNZ     TWT             ; LOOP UNTIL NOT BUSY
97          JMP     TKDATA          ; GO READ DATA FROM DOU AND RETURN
98 ;
99
```

The second method for reading the Fluke 8375 is to rely on the sample rate set from the front panel controls and to wait until a full transition of the busy flag is observed. This guarantees that a previous reading is not mistakenly interpreted as a new reading.

```
100
101 ;
102 ; *** SUBROUTINE TO OBTAIN NEXT READING ***
103 ;
104 NXTRD:
105         IN      FLG             ; INPUT THE BUSY FLAG
106         ANI     00100000B       ; TEST PORT C BIT 5
107         JZ      NXTRD           ; LOOP UNTIL BUSY WITH NEXT READING
108 NXTWT:
109         IN      FLG             ; INPUT THE BUSY FLAG
110         ANI     00100000B       ; TEST PORT C BIT 5
111         JNZ     NXTWT           ; LOOP UNTIL NOT BUSY
112         JMP     TKDATA          ; GO READ DATA FROM DOU AND RETURN
113 ;
114
```

Notice that the loops beginning at NXTWT in the above program segment and at TWT in the previous program segment are identical. This suggests the possibility of some obvious code optimization that is omitted here for the sake of clarity.

There is one subroutine remaining to complete full utilization of the Fluke 8375 DOU capabilities. It is the subroutine to take data from the 8375 DOU. The schematic (Figure 3) shows that port 5 (B) bits PB4–PB0 are used to enable the DOU drivers. Data from the DOU includes:

- 5 decades of digits
- encoded range and overrange

- function: Volts DC, Volts AC, Ohms, Kilohms
- modifiers: Filter, Ext. Ref., Remote
- overload
- trigger

The function of this subroutine is to read five bytes of data from the 8375 DOU and place them in a RAM buffer on the iSBC 80/10A.

```
115
116 ;
117 ; *** SUBROUTINE TO TAKE DATA FROM 8375 DOU ***
118 ;
119 TKDATA:
120         LXI     H,RDBUF         ; LD BUFFER POINTER
121         MVI     A,0EFH          ; SETUP FIRST STROBE
122 TKO:
123         MOV     B,A             ; SAVE CURRENT STROBE
124         OUT     STB             ; STROBE DECADE PAIR
125         IN      DATIN           ; READ DATA
126         MOV     M,A             ; PLACE DATA INTO SBC 80/10 RAM
127         INX     H               ; INCREMENT BUFFER POINTER
128         MOV     A,B             ; RESTORE STROBE
129         RNC                     ; ROTATE TO NEXT STROBE POSITION
130         JC      TKO             ; LOOP UNTIL BIT 0 STROBE DONE
131         OUT     STB             ; DISABLE ALL STROBES
132         RET
133 ;
134
```

This completes the software required to service the Fluke 8375 DOU. The following code consists of a routine to display the data from the interface on the console output device and a short executive program to allow exercising of the driver subroutines.

The display subroutine takes 5 bytes of data from the RAM buffer in which the reading has been stored and prints them, 2 ASCII characters per 8-bit byte, on the console.

```
135
136 ;
137 ; *** SUBROUTINE TO DISPLAY READING BUFFER ON CONSOLE ***
138 ;
139 DISPLAY:
140         LXI     H,RDBUF         ; LD BUFFER POINTER
141         MVI     D,5             ; INITIALIZE COUNTER
142 DISPO:
143         MOV     A,M             ; LD NEXT BYTE FROM BUFFER
144         CALL    NMOUT           ; CALL SBC 80/10 MONITOR SUBROUTINE
145                                 ;  TO DISPLAY ACCUMULATOR CONTENTS
146         INX     H               ; INCREMENT BUFFER POINTER
147         DCR     D               ; DECREMENT COUNTER
148         JNZ     DISPO           ; LOOP FOR 5 DISPLAY BYTES
149         RET
150 ;
151
```

Operator Interface. The short executive program provides a tool for the purposes of exercising the 8375 DOU driver subroutines. The executive begins by calling the initialization subroutine and then continues on to prompt the operator with a '>' on the console. At that point the operator may enter one of the following characters, causing the program to execute the specified subroutine:

	SUBR	DESCRIPTION
T	ETRIG	Enable external trigger
I	DTRIG	Disables external trigger
E	EPOS	Enable programmed timeouts
D	DPOS	Disable programmed timeouts
N	NXTRD	Next reading
S	TRGR	Trigger and get a reading
X	DISPLAY	Display reading buffer

After the operator has entered a command character, the program obtains the address of the subroutine to be executed and proceeds to set up a return address on the stack. This technique allows a load program counter instruction (PCHL) to be used to enter the subroutine and a return instruction (RET) to resume execution of the executive.

```
152
153 ;
154 ; *** SIMPLE EXECUTIVE EXERCISE PROGRAM ***
155 ;
156 START:
157         LXI     SP,STACK        ; SETUP STACK POINTER
158         CALL    INIT            ; INITIALIZE THE SBC 80/10 8255#2
159 EXEC:
160         CALL    CROUT           ; EXEC ENTRY POINT - PRINT <CR><LF>
161         MVI     C,'>'           ; C LOADED WITH PROMPT CHARACTER
162         CALL    CO              ; CONSOLE OUTPUT
163         CALL    GETCH           ; GET CMND CHAR, MASK OFF PARITY
164         CALL    CO              ; PRINT THE CHARACTER ON THE CONSOLE
165         MOV     A,C             ; PUT CHARACTER BACK INTO THE ACCUM
166         LXI     B,NCMDS         ; C CONTAINS LOOP AND INDEX COUNT
167         LXI     H,CTAB          ; HL POINTS TO CMND TABLE
168 EXECO:
169         CMP     M               ; COMPARE TABLE ENTRY AND CHARACTER
170         JZ      EXEC1           ; BRANCH IF EQUAL - CMND RECOGNIZED
171         INX     H               ; ELSE, INCREMENT TABLE POINTER
172         DCR     C               ; DECREMENT LOOP COUNT
173         JNZ     EXECO           ; BRANCH IF NOT AT TABLE END
174         JMP     EXEC            ; ELSE, CMND ILLEGAL - IGNORE IT
175 EXEC1:
176         LXI     H,CADR          ; LD ADR OF TABLE OF CMND SUBRS
177         DAD     B               ; ADD WHAT IS LEFT OF LOOP COUNT
178         DAD     B               ;  - EACH ENTRY IN CADR IS 2 BYTES
179         MOV     A,M             ; GET LSP OF ADR OF TABLE ENTRY TO A
180         INX     H               ; POINT TO NXT BYTE IN TABLE
181         MOV     H,M             ; GET MSP OF ADR OF TABLE ENTRY TO H
182         MOV     L,A             ; PUT LSP OF ADR OF TABLE ENTRY TO L
183         LXI     D,EXEC          ; SETUP RETURN ADR ON THE STACK
184         PUSH    D
185         PCHL                    ; NEXT INSTR COMES FROM CMND SUBR
186 ;
187
```

The command and address tables as well as the reading buffer follow to complete the application software.

```
188
189 ;
190 ;   COMMAND AND ADDRESS TABLES
191 ;
192 CTAB:
193         DB      'XSNDEIT'

194 NCMDS   EQU     $-CTAB          ; NUMBER OF VALID COMMANDS
195 ;
196 CADR:
197         DW      0
198         DW      ETRIG
199         DW      DTRIG
200         DW      EPOS
201         DW      DPOS
202         DW      NXTRD
203         DW      TRGR
204         DW      DISPLAY
205 ;
206 ;   READING BUFFER AND STACK SPACE
207 ;
208 RDBUF:
209         DS      5               ; READING BUFFER
210 ;
211
212
213         END     START           ; TRANSFER ADDRESS IS TO START
```

SUMMARY/CONCLUSIONS

This instrumentation control application has been presented to demonstrate the simple techniques used to apply the iSBC 80/10A to the task of interfacing instrumentation. A natural extension of this example would include the control of the Fluke 8375 RCU, as well as the control of many additional instruments to build a small ATE system.

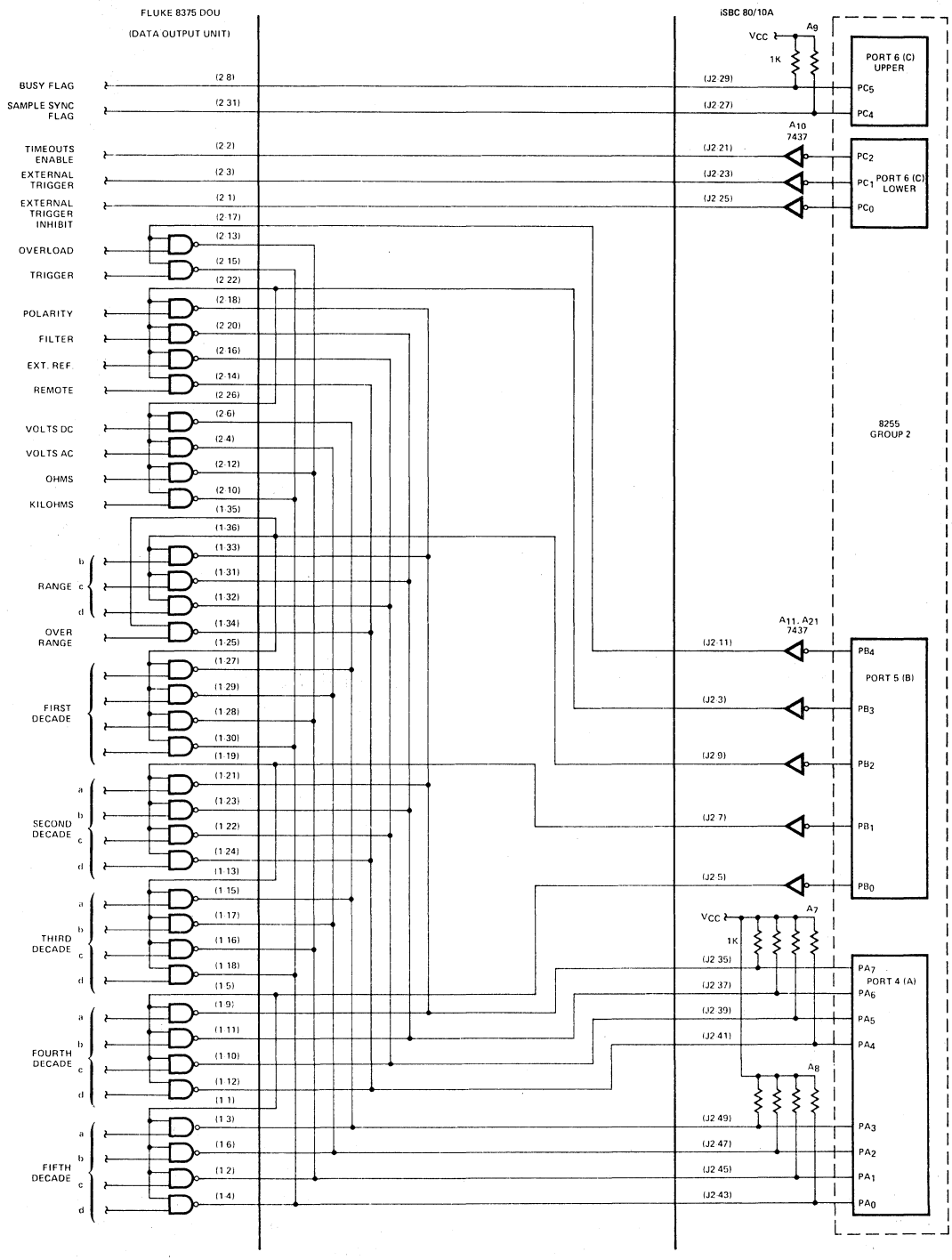


Figure 3. Interface Schematic

COMMUNICATION

A diverse range of single board computer applications exists in the field of communication. The increase in distributed processing generates requirements for self-contained computers to control elements of a communication system, increasing both the throughput and reliability.

There are many situations that necessitate monitoring and controlling a system from a remote site. Typical examples are systems that cover large geographic areas or systems in dangerous environments for human operators. If the object system, which provides the actual parallel inputs and outputs to the plant, is far from the controlling system, you can lower costs by reducing the number of interconnecting wires via the addition of multiplexers to both systems. In the extreme (and often desirable) case of reducing the interconnects to an absolute minimum, all communication between the systems takes place on a single serial data link. If large distances are involved, this link can be standard telephone wires. For moderate distances, the link can be a single twisted pair. In either case, the equipment used to interface the object system to the serial link is called a supervisory control and data acquisition (SCADA) terminal.

The decision to replace a multitude of interconnects with a SCADA terminal is largely economic. Cables and their associated drivers and receivers can represent a significant part of the total cost of a factory automation project, particularly if an electrically noisy environment requires the use of shielded cables. Any potential savings in cabling must, of course, compensate for the additional cost incurred by adding the SCADA terminal to the system.

Communication Application Example

A SCADA terminal demonstrates an industrial communication application of the iSBC 80/10A. The Intel® 8251 USART provides the serial communication link and the two Intel 8255 Programmable Parallel I/O devices provide 48 parallel lines for the object system. A block diagram of a SCADA terminal is shown in Figure 4.

The task of the software in this SCADA terminal example is two-fold. First, it must continually scan its parallel inputs, transmitting the status of those lines in a bit serial mode using the USART. And second, it receives bit serial data from the USART which is to be used to update the parallel outputs. Thus, a major portion of the software deals with

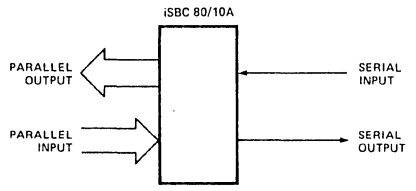


Figure 4. SCADA Terminal Block Diagram

the communications protocol on the serial data lines.

Communications Protocol. A communication protocol is an agreement between communications users that defines the record formats used for data transmissions. The protocol selected for this SCADA terminal application provides the following features:

1. A readable character set to simplify the human interface.
2. Error detection by means of a checksum.
3. Each record specifies the number of data bytes in the record and the initial port number.

Despite its value for human interface, the ASCII character set has problems representing 8-bit binary values, since the high-order bit is not used. Therefore, each binary value is treated as two 4-bit hexadecimal values. Because hexadecimal numbers fall in the range 0–9 and A–F, they can be represented as ASCII characters. However, this representation requires twice as many bytes as a pure binary format.

Record Format. The information encoded into the ASCII hexadecimal format is grouped to form records. Each record has a record mark to flag the beginning of the record, a number of ports specification (record length), destination output start port number, the data field itself, and a checksum.

The record format described below is according to the fields in the record.

Record mark field: Byte 0

The ASCII code for a colon (:) is used to signal the start of a record.

Number of ports field: Byte 1

The number of data bytes in the record is represented by a single ASCII hexadecimal digit in this field. This corresponds to the number of 8-bit

ports to which data will be output by the SCADA terminal in a parallel fashion. The maximum number of data bytes in a record is 15 (F in hexadecimal). A record length of zero is a special case and can be reserved for control information.

Port address field: Byte 2

The single ASCII hexadecimal digit in byte 2 gives the port number of the initial output port. The first data byte is output to the port indicated by the port address; successive bytes are output in successive port locations on the iSBC 80/10A or on expansion I/O boards.

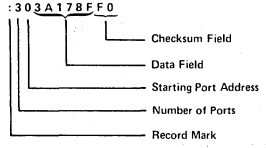
Data field: Bytes 3 to 3+2*(number of ports)–1

An 8-bit binary value is represented by two bytes containing the ASCII characters 0–9 or A–F, which represent a hexadecimal value between 0 and FF (0 and 255 decimal). The high-order digit is in the first byte of each pair.

Checksum field: Bytes 3+2*(number of ports) to 3+2*(number of ports)+1

The checksum field contains the ASCII hexadecimal representation of the two's complement of the 8-bit sum of the 8-bit bytes that result from converting each pair of ASCII hexadecimal digits to one byte of binary, from the number of ports field (the number of ports and port address constitute a pair) to and including the last byte of the data field. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the number of ports field to and including the checksum field, is zero.

Sample Hexadecimal format:



Design Approach Using a State Diagram. Before proceeding to examine the software used to implement the SCADA terminal, consider how the problem might have been approached with TTL logic rather than a microcomputer. The design would likely have been formulated with a state diagram to specify the transitions of a sequential state machine. The sequential-circuit operations would include decoding the serial input records and encoding the serial output records. An examination of the serial input record state diagram (Figure 5) is useful in understanding some of the procedures encountered later.

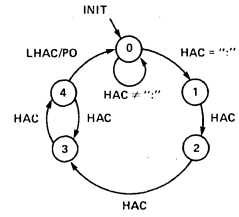


Figure 5. State Diagram

Notes: HAC = Hexadecimal ASCII character
LHAC = Last Hexadecimal ASCII character
PO = Parallel output

The receipt of an invalid HAC will cause a return to state 0.

The receipt of a colon at any time will produce a transition to state 1.

STATE		DESCRIPTION
0	=	record mark state
1	=	number of ports state
2	=	start port number state
3	=	high-order half of data byte state
4	=	low-order half of data byte state

State 0 is entered at the time of initialization. All state transitions occur when the next character is received. States 1, 2, and 3 are entered with the input of a colon (:), the number of ports and start port number, respectively. States 3 and 4 will cycle as required until all the high and low-order pairs of data have been input. The transition from state 4 to state 0 occurs when the last data byte has been received. If the checksum is correct, the parallel output latches are loaded with the data field of the record.

There are many references to the states contained in this diagram during the discussion of the software procedures. Thus, the state diagram is used as a "flowchart" for the software. As in the other examples in this application note, a textual description accompanies each segment of code. Intel's high-level programming language, PL/M-80, has been used to show the capability to program in a natural, algorithmic language which eliminates the need to manage register usage or memory allocation.

SCADA Terminal Program. The program begins with a comment, that is followed by the program segment label "SCADA". With resident PL/M-80, all programs are considered to be labelled blocks, or modules. This means that all PL/M programs must begin with a LABEL and a DO statement and end with an END statement.

```
                /*
                    INDUSTRIAL COMMUNICATION APPLICATION


                                SCADA  TERMINAL
                */

1               SCADA:
                DO;
```

All variables used in the program must be declared before they can be referred to by their identifiers. This is done by means of a DECLARE statement. In addition to the declaration of variables, macros are declared using the reserved word LITERALLY. These macros are expanded at compile time by textual substitution.

```
2   1       DECLARE
                SRL$IN$STATE BYTE,
                SRL$IN$PRT BYTE,
                SRL$IN$CNT BYTE,
                PRL$IN$STATE BYTE,
                PRL$IN$STRT$PRT BYTE,
                PRL$IN$NMB$PRTS BYTE,
                SRL$IN$PRL$OUT$BFR(3) BYTE,

                PRL$OUT$PRT$0 LITERALLY '0E5H',
                PRL$OUT$PRT$1 LITERALLY '0EAH',
                PRL$OUT$PRT$2 LITERALLY '0E8H',

                SRL$OUT$STATE BYTE,
                SRL$OUT$PRT BYTE,
                SRL$OUT$CNT BYTE,
                PRL$OUT$STATE BYTE,
                PRL$OUT$STRT$PRT BYTE,
                PRL$OUT$NMB$PRTS BYTE,
                SRL$OUT$PRL$IN$BFR(4) BYTE,

                PRL$IN$PRT$0 LITERALLY '0E4H',
                PRL$IN$PRT$1 LITERALLY '0E6H',
                PRL$IN$PRT$2 LITERALLY '0E9H',

                USART$CMD LITERALLY '0EDH',
                USART$IN LITERALLY '0ECH',
                USART$OUT LITERALLY '0ECH',
                USART$STATUS LITERALLY '0EDH',
                USART$MODE$INSTR LITERALLY '0CFH',
                USART$CMD$INSTR LITERALLY '025H',

                TXRDY LITERALLY '001H',
                RXRDY LITERALLY '002H',

                PPI$CWR$1 LITERALLY '0E7H',
                PPI$CWR$2 LITERALLY '0EBH',
                PPI$CWD$1 LITERALLY '080H',
                PPI$CWD$2 LITERALLY '09BH',

                TRUE LITERALLY 'OFFH',
                FALSE LITERALLY '000H',

                FOREVER LITERALLY 'WHILE TRUE',


                NEXT$BYTE BYTE,
                CHECKSUM BYTE;
```

8251 and 8255 Initialization. The INIT procedure sets up the 8251 and 8255's and initializes several variables. Interrupts are disabled to insure that no interrupts are serviced during the execution of the INIT procedure.

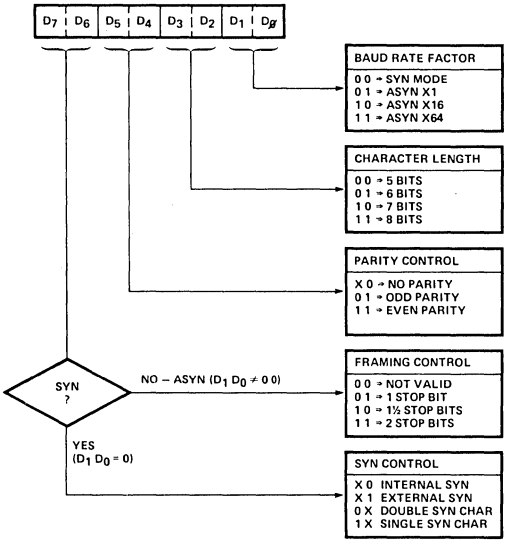
```
3   1       INIT: PROCEDURE;
4   2         DISABLE;
```

The serial input and serial output state counters are set to state 0. Port number 0 is the parallel input start port and 3 ports of data are input from the parallel ports for serial transmission.

```
5   2       SRL$IN$STATE = 0;
6   2       SRL$OUT$STATE = 0;
7   2       PRL$IN$STRT$PRT = 0;
8   2       PRL$IN$NMB$PRTS = 3;
```

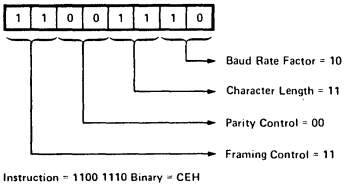
The Intel 8251 USART must be set up by loading it with mode and command instructions.

The mode instruction format is shown below:



The 8251 characteristics required by this SCADA terminal application include 9600 baud transmission and 8-bit characters. The parallel inputs of the 8255's are periodically scanned. The scanning frequency is determined by the baud rate and number of ports of data being transmitted. For example, the transmission of 3 ports of data requires 11 characters. At a baud rate of 9600 the approximate scan rate is 100 Hz.

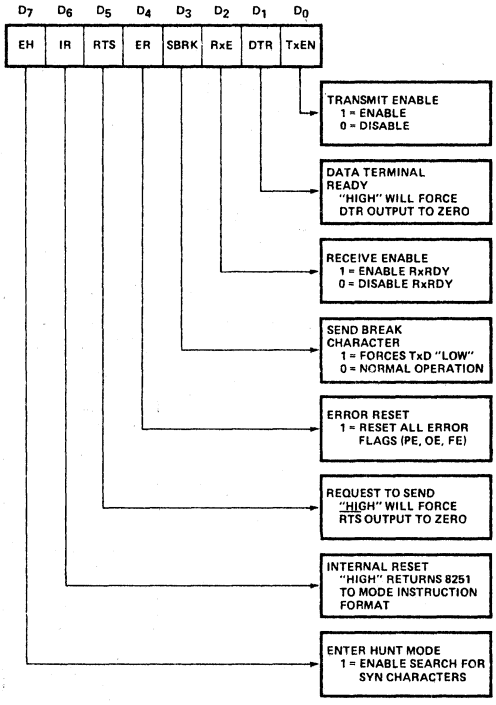
The following 8251 mode instruction is used:



Instruction = 1100 1110 Binary = CEH

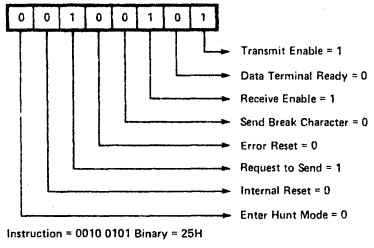
After the mode instruction is sent to the 8251, a command instruction is required to complete the 8251 initialization.

The command instruction format is shown below:



The command instruction enables the transmit and receive functions of the 8251.

The following command instruction is used:



Instruction = 0010 0101 Binary = 25H

Output instructions send the initialization commands to the 8251. Note that previously declared macros are used to literally replace the mnemonics in the following lines of code.

```
 9  2        OUTPUT(USART$CMD) = USART$MODE$INSTR;
10  2        OUTPUT(USART$CMD) = USART$CMD$INSTR;
```

Initialization of the 8255's is then done to set up the following configurations:

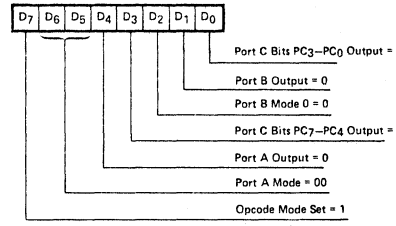
8255 #1

Port 1 (A)	Mode 0	Output
Port 2 (B)	Mode 0	Output
Port 3 (C)	Mode 0	Output

8255 #2

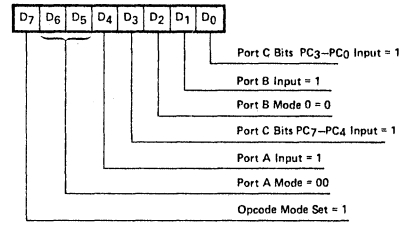
Port 4 (A)	Mode 0	Input
Port 5 (B)	Mode 0	Input
Port 6 (C)	Mode 0	Input

The following command instruction is used for the 8255 #1:



Mode Control Word = 1000 0000 Binary = 80H

The following command instruction is used for the 8255 #2:



Mode Control Word = 1001 1011 Binary = 9BH

The 8255 initialization commands are given in a similar manner to the 8251 commands.

```
11  2        OUTPUT(PPI$CWR$1) = PPI$CWD$1;
12  2        OUTPUT(PPI$CWR$2) = PPI$CWD$2;
```

The INIT procedure concludes by enabling interrupts.

```
13  2        ENABLE;
14  2    END INIT;
```

Conversion Procedures. Two conversion procedures are required in the program. The first procedure produces a hexadecimal ASCII character from a 4-bit binary value. A typed procedure has been used which returns a value of the type byte. It is called by using its name in an expression.

```
15  1    CHAR$CONV: PROCEDURE (CHAR) BYTE;

16  2       DECLARE CHAR BYTE;

17  2       CHAR = CHAR + '0';
18  2       IF CHAR > '9' THEN
19  2          CHAR = CHAR + 7;
20  2       RETURN CHAR;

21  2    END CHAR$CONV;
```

The second procedure produces a 4-bit binary value from a hexadecimal ASCII character. Because this procedure is used only when a hexadecimal ASCII character is expected, an illegal character (i.e., not a 0–9 or A–F) causes the serial input state counter to indicate state 0. This procedure is also typed. The NMB$CONV procedure emphatically illustrates the point that PL/M-80 performs unsigned arithmetic. Note that when the ASCII value for a zero is subtracted from the digit, NUM = NUM – '0'; a positive number is always produced, even if the value of NUM is less than '0'.

```
22  1    NMB$CONV: PROCEDURE (NMB) BYTE;

23  2       DECLARE NMB BYTE;

24  2       NMB = NMB - '0';
25  2       IF NMB > 9 THEN
26  2       DO;
27  3          IF (NMB > 16) AND (NMB < 23) THEN
28  3             NMB = NMB - 7;
            ELSE
29  3             SRL$IN$STATE = 0;
30  3       END;
31  2       RETURN NMB;

32  2    END NMB$CONV;
```

Parallel Input Procedure. A parallel input procedure is used to input data bytes from the 8255's. The data bytes are then transmitted by the bit serial output device. This procedure also computes the checksum for the serial output record. The checksum, TEMP2, is initialized to contain the parallel input number of ports and the start port, shifted to fit within a single byte. Each cycle of the iterative DO block adds the next data byte to the checksum and places the input data into the SRLOUTPRLINBFR array until the loop is complete. The checksum is then computed as the two's complement of the accumulated sum and also stored in the serial input parallel output buffer.

```
33  1    PARALLEL$IN: PROCEDURE;

34  2       DECLARE (TEMP1,TEMP2) BYTE;

35  2       TEMP2 = PRL$IN$NMB$PRTS * 16 + PRL$IN$STRT$PRT;

36  2       DO PRL$IN$STATE = PRL$IN$STRT$PRT TO
                              PRL$IN$STRT$PRT + PRL$IN$NMB$PRTS - 1;

37  3          DO CASE PRL$IN$STATE;

                   /* PRL IN PRT 0 */
38  4              TEMP1 = INPUT(PRL$IN$PRT$0);

                   /* PRL IN PRT 1 */
39  4              TEMP1 = INPUT(PRL$IN$PRT$1);

                   /* PRL IN PRT 2 */
40  4              TEMP1 = INPUT(PRL$IN$PRT$2);

41  4           END;

42  3       SRL$OUT$PRL$IN$BFR(PRL$IN$STATE) = TEMP1;
43  3       TEMP2 = TEMP2 + TEMP1;

44  3       END;

45  2       SRL$OUT$PRL$IN$BFR(PRL$IN$STRT$PRT + PRL$IN$NMB$PRTS) = -TEMP2;

46  2    END PARALLEL$IN;
```

Parallel Output Procedure. When a complete serial input record has been received and the checksum is correct, the transition from state 4 to state 0 is accompanied by the parallel output of the data from the data field of the serial input record. The parallel output starting port and the number of ports of data is contained in the input record and is thus used in directing the parallel output operation. An iterative DO block increments the PRLOUTSTATE index variable through the required ports and a DO CASE block selectively executes one of the OUTPUT statements for each cycle of the loop.

```
47  1    PARALLEL$OUT: PROCEDURE;

48  2       DECLARE TEMP BYTE;

49  2       DO PRL$OUT$STATE = PRL$OUT$STRT$PRT TO
                               PRL$OUT$STRT$PRT + PRL$OUT$NMB$PRTS - 1;

50  3          TEMP = SRL$IN$PRL$OUT$BFR(PRL$OUT$STATE);

51  3          DO CASE PRL$OUT$STATE;

                   /* PRL OUT PRT 0 */
52  4              OUTPUT(PRL$OUT$PRT$0) = TEMP;

                   /* PRL OUT PRT 1 */
53  4              OUTPUT(PRL$OUT$PRT$1) = TEMP;

                   /* PRL OUT PRT 2 */
54  4              OUTPUT(PRL$OUT$PRT$2) = TEMP;

55  4           END;
56  3       END;

57  2    END PARALLEL$OUT;
```

Serial Input and Output Procedures. The next two procedures contain the software implementations of the state diagram described previously. The processing during each state of the first procedure, the serial character input procedure, is described in the following text.

The procedure begins by reading a character from the 8251 and then converts the character into a 4-bit binary value using the number conversion procedure. The DO CASE block is the mechanism by which a program segment is selected to examine

the input character, provide the required outputs, and to specify the transition to the next state.

```
58  1   SERIAL$CHAR$IN: PROCEDURE;

59  2     DECLARE (CHAR,TEMP) BYTE;

60  2     CHAR = INPUT(USART$IN) AND 07FH;
61  2     TEMP = NMB$CONV(CHAR);

62  2     DO CASE SRL$IN$STATE;
```

State 0 is entered through the initialization process, at the completion of the processing of a serial input record, or when an invalid character has been received. The serial input state will remain 0 until a colon (:) is received, at which time a transition to state 1 is specified.

```
                  /* SRL IN STATE 0  =  RECORD MARK */
63  3             DO;
64  4               IF CHAR = ':' THEN
65  4                 SRL$IN$STATE = 1;
66  4             END;
```

The parallel output number of ports is obtained, the counter initialized, and a transition to state 2 is specified from state 1.

```
                  /* SRL IN STATE 1  =  NMB PRTS */
67  3             DO;
68  4               PRL$OUT$NMB$PRTS = TEMP;
69  4               SRL$IN$CNT = TEMP;
70  4               SRL$IN$STATE = 2;
71  4             END;
```

In state 2 the parallel output starting port number is obtained, the serial input port is initialized, the checksum is set to contain the parallel output number of ports and starting port, and a transition to state 3 is specified.

```
                  /* SRL IN STATE 2  =  STRT PRT */
72  3             DO;
73  4               PRL$OUT$STRT$PRT = TEMP;
74  4               SRL$IN$PRT = TEMP;
75  4               CHECKSUM = PRL$OUT$NMB$PRTS*16 + PRL$OUT$STRT$PRT;
76  4               SRL$IN$STATE = 3;
77  4             END;
```

In state 3 the high-order half of a data byte is obtained and shifted into the proper position of the NEXT$BYTE variable. A transition is specified to state 4.

```
                  /* SRL IN STATE 3  =  HI ORDER HALF DATA BYTE */
78  3             DO;
79  4               NEXT$BYTE = TEMP*16;
80  4               SRL$IN$STATE = 4;
81  4             END;
```

State 4 is the final state and requires more processing than the others. First, a whole byte of data is assembled by adding the low and high-order data halves, and then testing to determine if the checksum has been received. If so, and the checksum is correct, the parallel output procedure is executed. Once the entire serial input record has been received, a transition is specified to state 0 whether the checksum is correct or not. However, if the serial input count has not been exhausted, the assembled byte is placed into the serial input parallel output buffer and a transition back to state 3 is specified.

```
                  /* SRL IN STATE 4  =  LO ORDER HALF DATA BYTE */
82  3             DO;
83  4               NEXT$BYTE = NEXT$BYTE + TEMP;
84  4               CHECKSUM = CHECKSUM + NEXT$BYTE;
85  4               IF SRL$IN$CNT = 0 THEN
86  4               DO;
87  5                 IF CHECKSUM = 0 THEN
88  5                   CALL PARALLEL$OUT;
89  5                 SRL$IN$STATE = 0;
90  5               END;
                    ELSE
91  4               DO;
92  5                 SRL$IN$PRL$OUT$BFR(SRL$IN$PRT) = NEXT$BYTE;
93  5                 SRL$IN$PRT = SRL$IN$PRT + 1;
94  5                 SRL$IN$CNT = SRL$IN$CNT - 1;
95  5                 SRL$IN$STATE = 3;
96  5               END;
97  4             END;

98  3           END; /* END OF CASES */

99  2     END SERIAL$CHAR$IN;
```

The serial character output procedure is similar to the serial character input procedure. During state 0 the parallel inputs of the 8255's are stored in the serial output parallel input buffer for transmission.

```
100  1    SERIAL$CHAR$OUT: PROCEDURE;

101  2      DECLARE (CHAR,TEMP) BYTE;

102  2      CHAR = 0;

103  2      DO CASE SRL$OUT$STATE;

                /* SRL OUT STATE 0  =  RECORD MARK */
104  3          DO;
105  4            CHAR = ':';
106  4            CALL PARALLEL$IN;
107  4            SRL$OUT$STATE = 1;
108  4          END;

                /* SRL OUT STATE 1  =  NMB PRTS */
109  3          DO;
110  4            TEMP = PRL$IN$NMB$PRTS;
111  4            SRL$OUT$CNT = TEMP;
112  4            SRL$OUT$STATE = 2;
113  4          END;

                /* SRL OUT STATE 2  =  STRT PRT */
114  3          DO;
115  4            TEMP = PRL$IN$STRT$PRT;
116  4            SRL$OUT$PRT = TEMP;
117  4            SRL$OUT$STATE = 3;
118  4          END;

                /* SRL OUT STATE 3  =  HI ORDER HALF DATA BYTE */
119  3          DO;
120  4            TEMP = SHR(SRL$OUT$PRL$IN$BFR(SRL$OUT$PRT),4);
121  4            SRL$OUT$STATE = 4;
122  4          END;

                /* SRL OUT STATE 4  =  LO ORDER HALF DATA BYTE */
123  3          DO;
124  4            TEMP = SRL$OUT$PRL$IN$BFR(SRL$OUT$PRT) AND 0FH;
125  4            IF SRL$OUT$CNT = 0 THEN
126  4              SRL$OUT$STATE = 0;
                   ELSE
127  4            DO;
128  5              SRL$OUT$CNT = SRL$OUT$CNT - 1;
129  5              SRL$OUT$PRT = SRL$OUT$PRT + 1;
130  5              SRL$OUT$STATE = 3;
131  5            END;
132  4          END;

133  3        END; /* END OF CASES */

134  2        IF CHAR <> ':' THEN
135  2          CHAR = CHAR$CONV(TEMP);
136  2        OUTPUT(USART$OUT) = CHAR;

137  2      END SERIAL$CHAR$OUT;
```

Interrupt Service Routine. The software in this SCADA terminal application example is interrupt driven. Interrupts, which occur when the transmitter of the 8251 is ready for another character or when the receiver has obtained a serial character, direct the execution of either the serial input

or output character procedures. The following procedure is entered when an interrupt occurs.

```
138  1    USART$INTERRUPT: PROCEDURE INTERRUPT 7;
139  2       DECLARE STATUS BYTE;
140  2       STATUS = INPUT(USART$STATUS);
141  2       IF (STATUS AND TXRDY) = TXRDY THEN
142  2          CALL SERIAL$CHAR$OUT;
143  2       IF (STATUS AND RXRDY) = RXRDY THEN
144  2          CALL SERIAL$CHAR$IN;
145  2    END USART$INTERRUPT;
```

Main Program. The function of the main program is rather simple. It calls the initialization routine and then loops "FOREVER." Notice that the other software is executed only when an interrupt occurs. Rather than loop idly while waiting for an interrupt, the "main program" could take advantage of excess CPU time by processing some other task.

```
                /***********
                MAIN$PROGRAM:
                ***********/
146  1          CALL INIT;
147  1          DO FOREVER;
148  2          END;


149  1    END;
```

SUMMARY/CONCLUSIONS

Further consideration should be given to error checking in the implementation of a SCADA terminal. A checksum has been used in this example which provides some error detection but no correction.

The industrial communication example in this application note has shown a SCADA terminal. Besides providing a convenient forum in which to explore the use of PL/M in an interrupt-driven environment, this application provides a realistic and almost fully-developed tool for the replacement of a multitude of parallel lines. Two such systems can be connected through the serial lines to provide a parallel to parallel transmission scheme as shown in Figure 6.

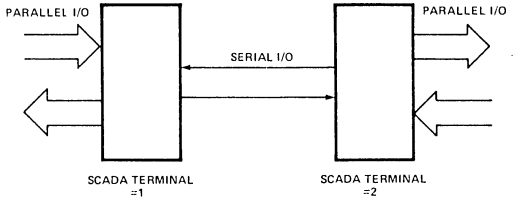


Figure 6. Two SCADA Terminals

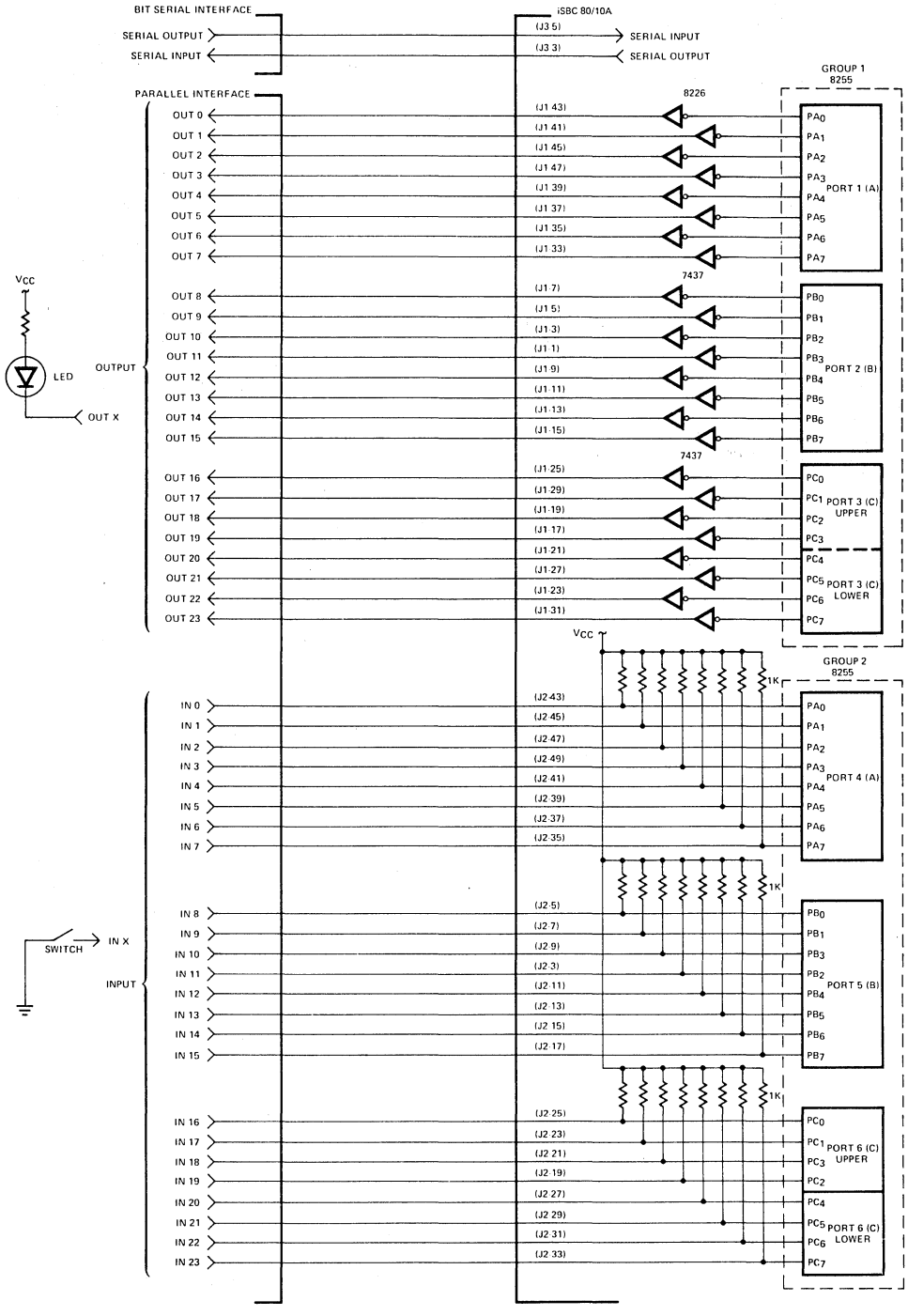


Figure 7. SCADA Terminal Schematic

PROCESS CONTROL

Many single board computers have already been applied in the field of process control. Some of the common denominators observed in these applications include the use of A/D and D/A peripheral boards, process monitoring functions such as servicing display panels for operator interaction, and alarm indicators.

Temperature Monitoring Application Example

A temperature monitoring system has been developed for the purposes of a process control application example. The single open loop system utilizes an A/D converter, a multiplexed display, switches for operator control, and two alarms. A block diagram of the operator's panel is shown in Figure 8 and a schematic in Figure 9.

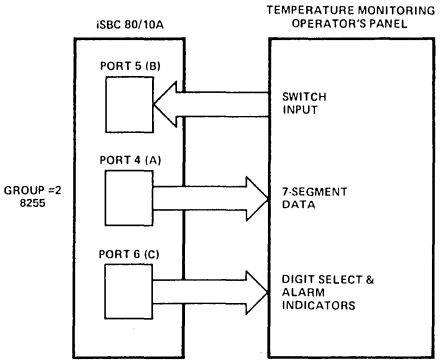


Figure 8. Operator's Panel Block Diagram

Operator's Panel. The operator's panel in this temperature monitoring system contains four 7-segment displays to show the temperature, two light emitting diodes (LEDs) that indicate alarm-low and alarm-high conditions, and six switches. The function of the switches is as follows:

Set Limit – controls whether the current temperature reading is to be displayed (off) or if upper/lower limits are to be set (on).

Set Hi Lo – when set limit is "on", this switch controls whether the low (off) or high (on) limit is to be displayed.

Digit Selects – these two switches control the selection of the digit of the limit which is to be modified. The four binary positions 00, 01, 10 and 11 correspond to the four 7-segment digits.

Leave It – controls whether the digit selected is to be incremented (off) or maintained at its current value (on). When this switch is "off" the digit selected is incremented every 512 ms until the operator turns the switch "on".

Enable Alarm – when set limit is "off" and the current temperature is displayed, this switch controls whether the action of the alarm indicators is to be enabled (on) or disabled (off).

The simple means used to set upper and lower temperature limits is similar to setting the time on a digital wrist watch.

The purpose of the software is to initialize the system and then to enter an endless loop which accumulates 16 readings, updates the displayed reading or handles limit setting, updates the display latches, waits 4 ms, and obtains an A/D reading.

Temperature Monitoring Program. This application example has been coded in Intel's resident PL/M-80 language.

```
                /*
                    PROCESS CONTROL APPLICATION

                            OPEN LOOP

                        TEMPERATURE MONITOR
                */
1               TEMPERATURE$MONITOR:
                DO;
```

The declaration statement includes some dimensioned variables with INITIAL attributes. They provide data strobe positions, a table of bit patterns to convert BCD data to 7-segment data, and a table of the powers of 10 for binary to BCD conversions.

```
2  1    DECLARE
            READING ADDRESS,
            DIGITS(4) BYTE INITIAL (80H,40H,20H,10H),
            BCDTO7SEG(11) BYTE INITIAL (3FH,06H,5BH,4FH,66H,
                        6DH,7CH,07H,7FH,67H,0),
            TENS(4) ADDRESS INITIAL (1000,100,10,1),
            DIGIT$DATA(4) BYTE,
            NXT$DIGIT BYTE,
            UPDATE$COUNT BYTE,
            SET$COUNT BYTE,
            LIMIT(2) ADDRESS,
            ACCUM$RDNG ADDRESS,

            CWR LITERALLY '0EBH',
            SLCT LITERALLY '0EAH',
            SEGS LITERALLY '0E3H',
            SWTS LITERALLY '0E9H',
            SETUP$PORTS LITERALLY '082H',

            SET$LIMIT LITERALLY '001H',
            SET$HI$LO LITERALLY '002H',
            LEAVE$IT LITERALLY '010H',
            DIGIT$SLCT LITERALLY '00CH',
            ENABLE$ALARM LITERALLY '020H',
            SET$ALARM$LO LITERALLY '001H',
            SET$ALARM$HI LITERALLY '003H',
            RESET$ALARM$LO LITERALLY '000H',
            RESET$ALARM$HI LITERALLY '002H',

            TRUE LITERALLY '0FFH',
            FOREVER LITERALLY 'WHILE TRUE';
```

The analog to digital conversion procedure has been coded in assembly language and is not included in this application note. It is declared as an external typed procedure with no arguments and returns a value of the type address. The value returned is the current temperature. The ATOD procedure is linked later in a step to produce an absolute load module of the entire program.

```
3  1    ATOD: PROCEDURE ADDRESS EXTERNAL;
4  2    END ATOD;
```

Bit set/reset functions of the 8255 have been used to control the alarm-low and high output bits. Use of this function allows individual bits to be controlled without affecting others of port C which are concurrently selecting the digit to be multiplexed on the display.

```
5  1    RESET$ALARMS: PROCEDURE;
6  2       OUTPUT(CWR) = RESET$ALARM$LO;
7  2       OUTPUT(CWR) = RESET$ALARM$HI;
8  2    END RESET$ALARMS;
```

The following procedure is used to initialize the 8255 and several program variables.

```
9  1    INIT: PROCEDURE;
10 2       OUTPUT(CWR) = SETUP$PORTS;
11 2       CALL RESET$ALARMS;
12 2       NXT$DIGIT = 0;
13 2       UPDATE$COUNT = 0;
14 2       SET$COUNT = 7;
15 2       READING = 0;
16 2       ACCUM$RDNG = 0;
17 2       LIMIT(0) = 0000;
18 2       LIMIT(1) = 9999;
19 2    END INIT;
```

A multiplexed display is controlled by the software. Two ports of an 8255 are required for this function shown in Figure 9. The first output port holds the data which drives the four 7-segment displays in parallel. The second output port contains four strobes, each going to a separate common cathode of one of the 7-segment displays.

The update display procedure begins by blanking 7-segment data in the output port. This step avoids shadows that would be produced if the data for the next digit position were loaded prior to updating the strobe. The strobe is then advanced, retaining the alarm bits that occupy other bits of the same output port. Note that an output configured 8255 port can be read with an 8080A INPUT instruction to determine the currently latched output data. The BCD data is obtained from the next digit position of the DIGIT$DATA array and used as a subscript into a table of BCDTO7SEG data. The 7-segment data is also output to the 8255 port in the same statement. The procedure concludes by advancing the NXT$DIGIT pointer.

```
20 1    DISPLAY$UPDATE: PROCEDURE;
21 2       OUTPUT(SEGS) = 0;
22 2       OUTPUT(SLCT) = (DIGITS(NXT$DIGIT) OR (INPUT(SLCT) AND 03H));
23 2       OUTPUT(SEGS) = BCDTO7SEG(DIGIT$DATA(NXT$DIGIT));
24 2       NXT$DIGIT = (NXT$DIGIT+1) AND 03H;
25 2    END DISPLAY$UPDATE;
```

Binary to BCD Conversion. Binary data from the A/D converter must be converted to BCD before it can be used by the DISPLAY$UPDATE procedure to show the current temperature reading. The BINTOBCD procedure performs this conversion operation.

```
26 1    BINTOBCD: PROCEDURE;
27 2       DECLARE (BCD,I) BYTE;
28 2       DO I = 0 TO 3;
29 3          BCD = 0;
30 3          DO WHILE READING >= TENS(I);
31 4             READING = READING - TENS(I);
32 4             BCD = BCD + 1;
33 4          END;
34 3          DIGIT$DATA(I) = BCD;
35 3       END;
36 2    END BINTOBCD;
```

BCD to Binary Conversion. The reverse conversion process is also needed. That is, BCD data must be converted to binary. This procedure is used to take limits, which are set by manipulating BCD digits, and convert them to binary data for use in testing against current temperature readings. Based variables have been used in this procedure to allow access to the actual variables used as arguments in the calling program.

```
37 1    BCDTOBIN: PROCEDURE (BCD$ARRAY$ADR,BIN$DATA$ADR);
38 2       DECLARE
            (BCD$ARRAY$ADR,BIN$DATA$ADR) ADDRESS,
            (BCD$ARRAY BASED BCD$ARRAY$ADR) (4) BYTE,
            (BIN$DATA BASED BIN$DATA$ADR) ADDRESS,
            I BYTE;
39 2       BIN$DATA = 0;
40 2       DO I = 0 TO 3;
              /* BIN$DATA = BIN$DATA*10 + BCD$ARRAY(I) */
41 3          BIN$DATA = SHL(BIN$DATA,1) + SHL(BIN$DATA,3) + BCD$ARRAY(I);
42 3       END;
43 2    END BCDTOBIN;
```

Updating the Display. The UPDATE procedure is entered each time 16 readings have been taken from the A/D converter. The UPDATE$COUNT is reset and the operator switches are input to control the execution path through the procedure. The accumulated reading, which is the total of 16 A/D readings, is divided by 16 to obtain an average reading. Then the accumulated reading is zeroed.

```
44  1   UPDATE: PROCEDURE;

45  2      DECLARE (SWT$FLG,HI$LO,DIGIT) BYTE;

46  2      UPDATE$COUNT = 15;
47  2      SWT$FLG = INPUT(SWTS);
48  2      READING = SHR(ACCUM$RDNG,4);
49  2      ACCUM$RDNG = 0;
```

Setting Limits. If the set limit switch is ON, the limits are to be dealt with instead of testing and displaying the current temperature reading. The alarms are reset during limit setting. The specified limit is converted to BCD and then the Leave-It switch is tested to see if the digit selected is to be incremented or held constant.

```
50  2      IF (SWT$FLG AND SET$LIMIT) = SET$LIMIT THEN
51  2      DO;
52  3         CALL RESET$ALARMS;
53  3         HI$LO = SHR((SWT$FLG AND SET$HI$LO),1);
54  3         READING = LIMIT(HI$LO);
55  3         CALL BINTOBCD;
56  3         IF (SWT$FLG AND LEAVE$IT) <> LEAVE$IT THEN
```

Another counter is used to control digit incrementing. Its purpose is to control the rate at which the selected digit is to be incremented. The major loop in the program has a 4-millisecond delay. Thus, 16 A/D conversions require a period of 64 ms which provides an update frequency of 16 readings per second. This is too fast to accurately select a desired digit which is being incremented. SET$COUNT insures eight update periods (512 ms) between each increment. After the digit has been incremented, the BCD limit value is converted back to binary to set the respective limit. This concludes the action taken when setting limits.

```
57  3            DO;
58  4               IF SET$COUNT = 0 THEN
59  4               DO;
60  5                  SET$COUNT = 7;
61  5                  DIGIT = SHR((SWT$FLG AND DIGIT$SLCT),2);
62  5                  IF DIGIT$DATA(DIGIT) = 9 THEN
63  5                     DIGIT$DATA(DIGIT) = 0;
                       ELSE
64  5                     DIGIT$DATA(DIGIT) = DIGIT$DATA(DIGIT) + 1;
65  5                  CALL BCDTOBIN(.DIGIT$DATA,.LIMIT(HI$LO));
66  5               END;
                    ELSE
67  4                  SET$COUNT = SET$COUNT - 1;
68  4            END;
69  3         END;
```

Testing the Averaged Reading. If the set limit switch is OFF, then the averaged reading is to be tested and displayed. The averaged reading is converted to BCD and then a test is performed to determine whether the reading is to be compared with the upper and lower limits.

```
                  ELSE
70  2         DO;
71  3            CALL BINTOBCD;
72  3            IF (SWT$FLG AND ENABLE$ALARM) = ENABLE$ALARM THEN
```

The reading is compared with both the upper and lower limits if the alarms have been enabled. The results of the tests are used to set and reset the corresponding alarm output bits.

```
73  3            DO;
74  4               IF HEADING < LIMIT(0) THEN
75  4                  OUTPUT(CWR) = SET$ALARM$LO;
                    ELSE
76  4                  OUTPUT(CWR) = RESET$ALARM$LO;

77  4               IF HEADING > LIMIT(1) THEN
78  4                  OUTPUT(CWR) = SET$ALARM$HI;
                    ELSE
79  4                  OUTPUT(CWR) = RESET$ALARM$HI;
80  4            END;
```

If the alarms are not enabled, both the alarms are reset to the "off" condition.

```
                  ELSE
81  3               CALL RESET$ALARMS;
82  3         END;

83  2      END UPDATE;
```

Main Program. The main program is shown below. Its purpose is to initialize the system and then to cycle, continuously executing the code previously described.

```
                  /*************
                  MAIN$PROGRAM:
                  *************/
84  1      CALL INIT;

85  1      DO FOREVER;

86  2         ACCUM$RDNG = ACCUM$RDNG + READING;

87  2         IF UPDATE$COUNT = 0 THEN
88  2            CALL UPDATE;
                 ELSE
89  2            UPDATE$COUNT = UPDATE$COUNT - 1;

90  2         CALL DISPLAY$UPDATE;
91  2         CALL TIME(40);
92  2         READING = ATOD;

93  2      END;


94  1      END;
```

SUMMARY/CONCLUSIONS

The goal of this application example is to demonstrate some of the common functions required for process control systems. Rather than show a small portion of a larger, more complex problem, this example was chosen because it presents a complete solution to a smaller problem. In summary, refreshing a multiplexed display was shown. Conversion procedures for binary to BCD and BCD to binary were used. A simple technique, in terms of hardware requirements, was used to enter lower and upper test values. And, limits testing was done, providing alarm indicators.

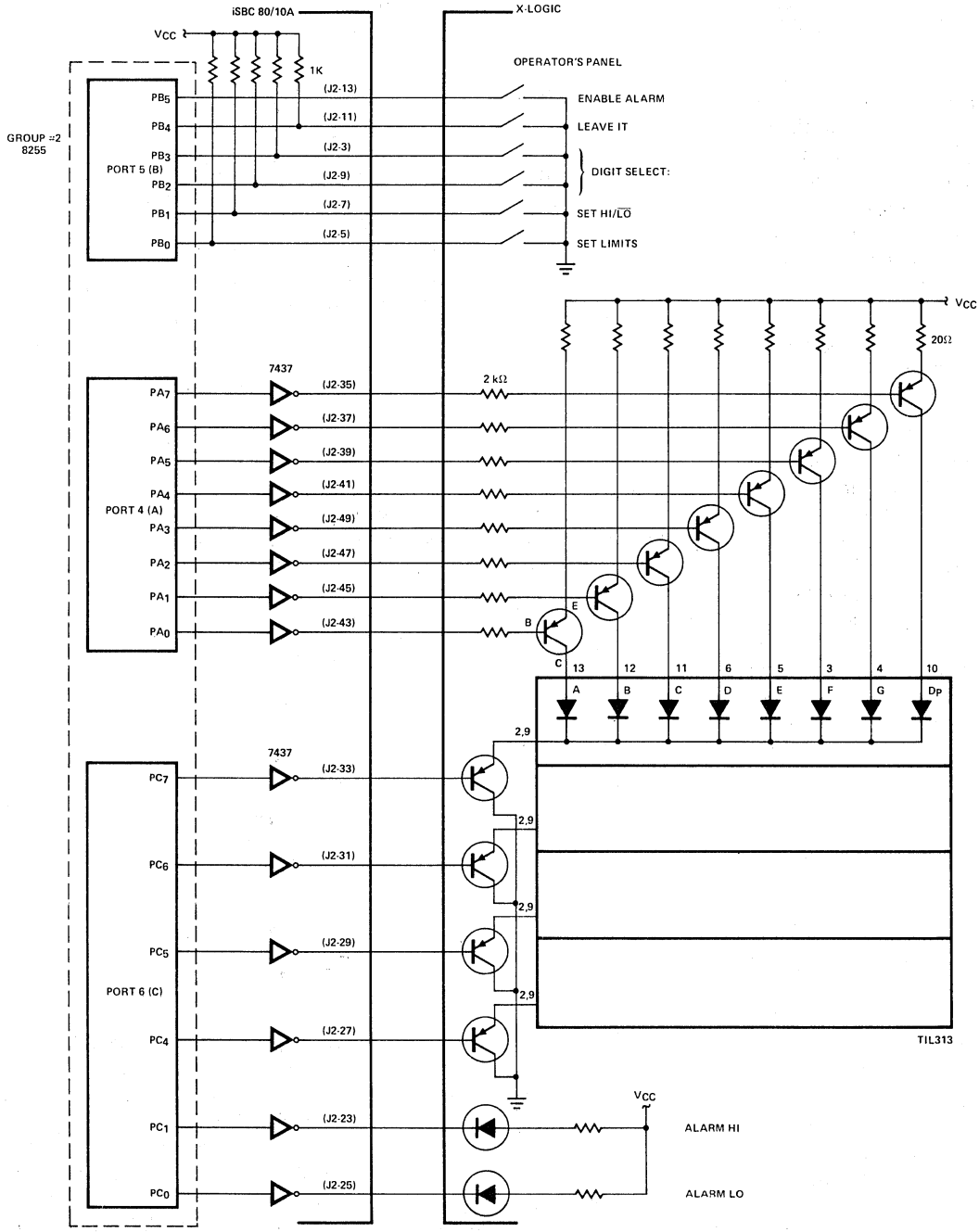


Figure 9. Operator's Panel Schematic

I/O DEVICE CONTROLLER

Peripheral processors have become common elements in computer systems of all sizes and capabilities. The purpose of such a processor is to relieve a central processor from the burden of handling I/O devices. In effect, it is a form of distributed processing. The iSBC 80/10A can be used as a peripheral processor and/or as an I/O device controller. In such a capacity it can significantly reduce the amount of hardware required to interface peripherals. Because the iSBC 80/10A controls only I/O, it is of little consequence that it must do a great deal of detail work that otherwise wastes the processing capability of a larger central processor.

Consider the activity of producing a listing on a line printer. The overhead in maintaining a program in the queue of a central processor which is producing data for a line printer can seriously impact system throughput. If, however, the program were to send the list to a disk file and then command a peripheral processor to take care of the printing, a significant improvement in system performance may be achieved. Printers represent one example of a large number of I/O devices that can be controlled by an iSBC 80/10A. Other devices include cassettes, magnetic tape drives, paper tape readers and punches, etc.

Character Printer Controller Application Example

The control of a Centronics 306 character printer is used as an I/O device controller application example. This example shows the interrupt capability of mode 1 8255 operation. A block diagram of the printer controller is shown in Figure 10 and a schematic in Figure 11.

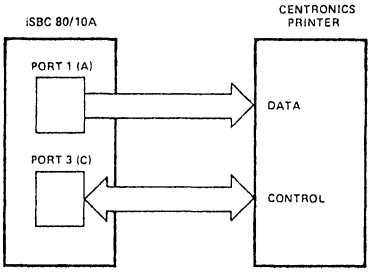


Figure 10. Printer Controller Block Diagram

When the mode 1 or mode 2 configuration is used, software is generally required to support interrupts used in conjunction with handshaking operations. Software routines written for an interrupt driven environment tend to be more complex than status driven routines. The added complexity is because interrupt-driven systems are constructed such that other software tasks are run while the I/O transaction is in progress. A software routine that controls a peripheral device is generally referred to as a device driver. One method of implementing an interrupt-driven device driver is to partition the device driver into a "command processor" and an "interrupt service routine." The command processor is the module that validates and initiates user program requests to the device driver. A common method of passing information between the various software programs is to have the requesting routine provide a device control block in memory. The device control block used in this application example is shown in Table 2.

Table 2. Printer Software Control Block

NAME	POSITION	DEFINITION
Status	Byte 0	A 1-byte field which defines the completion status of an I/O. 00 = Good completion 01 = Error – command already in progress.
Buffer Address	Byte 1, 2	Pointer to the start of the data to print.
Character Count	Byte 3	Count of the number of characters to print.
Character Transferred Count	Byte 4	The number of characters transferred.
Completion Address	Byte 5, 6	Address of a user supplied routine which will be called after the I/O has been performed.

The command processor validates the transaction and initiates the operation described by the control block. Control is then returned to the requester so that other processing may proceed. The interrupt service routine processes the remainder of the transaction.

Interrupt Service Routine Requirements. The interrupt service routine requires the following functions:

1. The state of the machine (registers, status, etc.) must be saved so that it may be restored after the interrupt is processed.
2. The source of the interrupt must be determined. The hardware may support a register which indicates the interrupting device, or the software may poll the device status registers.
3. Data must be passed to or from the device.
4. Control must be passed to the requesting routine at the completion of the I/O.
5. The state of the machine must be restored before returning to the interrupted program.

Printer Controller Program. The software for this application has been coded using Intel® 8080 Macro Assembly Language.

```
0 ;
1 ;*****
2 ;
3 ;       I/O DEVICE CONTROLLER APPLICATION
4 ;
5 ;               INTERRUPT DRIVEN
6 ;
7 ;               CHARACTER PRINTER
8 ;
9 ;*****
```

The following program equates are used to allow symbolic reference to the 8255 ports. Group #1 8255 on the iSBC 80/10A has been used because it will support mode 1 operation.

```
10 ;
11 ;*****
12 ;       PROGRAM EQUATES
13 ;*****
14 PORTA   EQU     0E4H    ; 8255 PORT A
15 PORTB   EQU     0E5H    ; 8255 PORT B
16 PORTC   EQU     0E6H    ; 8255 PORT C
17 CWR     EQU     0E7H    ; 8255 CONTROL WORD REGISTER
```

An initialization control word sent to the control word register of the 8255 will set up the desired configuration.

```
18 ;
19 ;*****
20 ;
21 ;       INITIALIZATION CONTROL WORD
22 ;
23 ;               USED TO CONFIGURE THE 8255 AS FOLLOWS:
24 ;
25 ;                       PORT A - OUTPUT MODE 1
26 ;                       PORT B - INPUT MODE 0 (NOT USED)
27 ;                       PORT C LOWER - OUTPUT
28 ;
29 ;*****
30 ICW     EQU     10101010B       ; INITIALIZATION CONTROL WORD
31 ;*****
```

The bit set/reset capability of the 8255 is used to control the strobe to the printer and to enable/disable interrupts from the 8255.

```
32 ;       SET/RESET CONTROL WORDS
33 ;*****
34 STBON   EQU     00000001B       ; SET STROBE
35 STBOF   EQU     00000000B       ; RESET STROBE
36 ;*****
37 ;       8255 ENABLE/DISABLE INTERRUPT CONTROL WORDS
38 ;*****
39 IEN     EQU     00001101B       ; ENABLE INTERRUPTS
40 IDN     EQU     00001100B       ; DISABLE INTERRUPTS
41 ;*****
```

Device status, control block, and completion equates are shown below.

```
42 ;       DEVICE STATUS EQUATES
43 ;*****
44 LPBSY   EQU     080H    ; BUFFER FULL (LINE PRINTER BUSY)
45 INTRA   EQU     08H     ; INTERRUPT REQUEST
46 ;*****
47 ;       CONTROL BLOCK EQUATES
48 ;*****
49 CBST    EQU     00H     ; STATUS BYTE
50 CBUF    EQU     01H     ; BUFFER ADDRESS
51 CBCC    EQU     03H     ; CHARACTER COUNT
52 CBCT    EQU     04H     ; CHARACTER TRANSFERED COUNT
53 CBCMP   EQU     05H     ; COMPLETION SERVICE ADDRESS
54 ;*****
55 ;       COMPLETION STATUS EQUATES
56 ;*****
57 STGD    EQU     00H     ; GOOD COMPLETION
58 STE1    EQU     01H     ; ERROR - COMMAND ALREADY IN PROGRESS
59 ;*****
```

There are two origin statements in this program. The first origin at 38 hexadecimal is the entry point used when an interrupt is generated by the 8255. A jump instruction to the printer interrupt routine is stored at that location. The second origin at 3000 hexadecimal is the address where the rest of the code will be located.

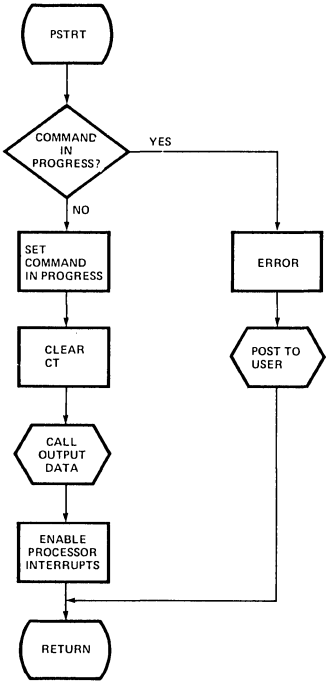
```
60 ;       RESTART 7 ENTRY POINT
61 ;*****
62         ORG     0038H
63         JMP     PINT
64 ;*****
65 ;       PROGRAM ORIGIN
66 ;*****
67         ORG     3000H
68 ;*****
```

An initialization subroutine issues the mode control word to the 8255 control word register after reset of the device. The printer strobe must then be disabled.

```
69 ;
70 ;       INITIALIZATION ROUTINE
71 ;
72 ;       A,H,L REGISTERS MODIFIED
73 ;
74 ;*****
75 INIT:
76         MVI     A,ICW   ; GET MODE CONTROL WORD
77         OUT     CWR     ; OUTPUT TO CONTROL WORD REGISTER
78         MVI     A,STBON ; GET SET DATA STROBE CONTROL WORD
79         OUT     CWR     ; SET DATA STROBE (LOW TRUE SIGNAL)
80         RET             ; RETURN TO CALLER
81
```

The command processor is started by the user routine through a subroutine call to PSTRT, with the address of the control block in the D and E registers. The command processor insures that an I/O operation is not already in progress, starts the I/O, enables interrupts, and returns to the caller so that other processing may proceed.

The flowchart and listing for the command processor are shown below.

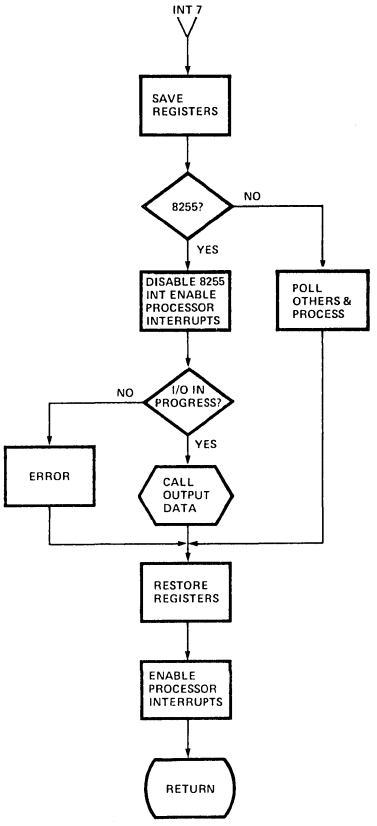



```
82
83 ;*****
84 ;
85 ;       COMMAND PROCESSOR
86 ;
87 ;               INPUTS: CONTROL BLOCK ADDRESS IN D AND E REGISTERS
88 ;
89 ;               OUTPUTS: START I/O OR ERROR STATUS IN CONTROL BLOCK
90 ;
91 ;       A,H,L REGISTERS MODIFIED
92 ;
93 ;*****
94 PSTRT:
95          LDA     PIPRG+1 ; GET PRINT IN PROGRESS BLOCK ADDRESS
96          ANA     A       ; SEE IF ZERO (PRINT IN PROGRESS)
97                          ; IF BLOCK ADDRESS NOT EQUAL TO ZERO THEN
98                          ; PRINT IN PROGRESS
99          JNZ     PSTE    ; IF YES - BRANCH TO ERROR
100         XCHG
101         SHLD    PIPRG   ; SAVE CONTROL BLOCK ADDRESS
102         XCHG
103         LXI     H,CBCT  ; GET INDEX TO CT
104         DAD     D       ; COMPUTE ADDRESS OF CT
105         MVI     M,00H   ; CLEAR CT
106         CALL    PDATA   ; START I/O
107         EI              ; ENABLE PROCESSOR INTERRUPTS
108         RET             ; RETURN TO CALLER
109 ;*****
110 ;       ERROR - TRANSACTION ALREADY IN PROGRESS
111 ;*****
112 PSTE:
113         MVI     A,STE1  ; GET ERROR STATUS CODE
114         JMP     POST    ; PASS CONTROL TO COMPLETION ROUTINE
115
```

Interrupt Processing. When the 8255 generates an interrupt, the interrupt request is serviced by the 8080A CPU. The CPU disables processor interrupts and then executes the instruction at location 38 hexadecimal, which is a jump to the interrupt service routine. The interrupt service routine saves the processor state and polls the 8255 to determine the source of the interrupt. Once the interrupting device is identified, the printer output data routine is called. After the entire buffer has been printed, the interrupt service routine passes control to the user-supplied completion routine. Before returning from the interrupt, the state of the processor is restored.

There are a number of error conditions which may occur, such as an interrupt from a device that does not have an active control block, or an interrupt when polling establishes that no device requires service. Neither of these errors should occur, but if they do, the driver should perform in a consistent fashion. The recovery routines implemented to handle these interrupt error conditions are determined by the environment of the particular application.

The flowchart and listing for the printer interrupt service routine are shown below.



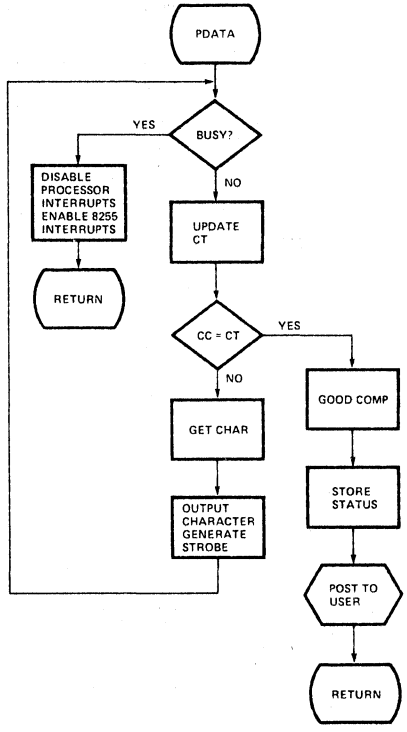

```
116
117 ;*****
118 ;       PRINTER INTERRUPT SERVICE ROUTINE
119 ;       ALL REGISTERS SAVED AND RESTORED
120 ;*****
121 PINT:
122         PUSH    PSW     ; SAVE PSW
123         PUSH    B       ; SAVE REGISTER PAIR B AND C
124         PUSH    D       ; SAVE REGISTER PAIR D AND E
125         PUSH    H       ; SAVE REGISTER PAIR H AND L
126 ;*****
```

```
127 ;       POLL INTERRUPT SOURCE - SEE OF 8255
128 ;*****
129         IN      PORTC   ; GET STATUS OF DEVICE
130         ANI     INTRA   ; SEE IF INT
131         JZ      PPOLL   ; NO -BRANCH TO POLL OTHER DEVICES IF ANY
132         MVI     A,IDN   ; GET 8255 INT DISABLE CONTROL WORD
133         OUT     CWR     ; DISABLE DEVICE INTERRUPTS
134         EI              ; ENABLE PROCESSOR INTERRUPTS
135         LHLD    PIPRG   ; GET CONTROL BLOCK ADDRESS
136         XRA     A       ; CLEAR A REG
137         CMP     H       ; SEE IF PRINT IN PROGRESS
138         JZ      PIER1   ; NO - BRANCH TO ERROR ROUTINE
139         XCHG
140         CALL    PDATA   ; PRINT DATA
141 ;*****
142 ;       RESTORE REGISTERS AND RETURN FROM INTERRUPT
143 ;*****
144 PRTN:
145         POP     H       ; RESTORE REGISTER PAIR H AND L
146         POP     D       ; RESTORE REGISTER PAIR D AND E
147         POP     B       ; RESTORE REGISTER PAIR B AND C
148         POP     PSW     ; RESTORE PSW AND A
149         EI              ; ENABLE PROCESSOR INTERRUPTS
150         RET             ; RETURN TO INTERRUPTED PROCESS
151 ;*****
152 ;       POLL OTHER DEVICES IF ANY
153 ;               IF NO OTHER DIVICES TO POLL - USER SUPPLIED ERROR
154 ;               RECOVERY ROUTINE.
155 ;*****
156 PPOLL:
157         JMP     PRTN    ; RETURN
158 ;*****
159 ;       ERROR - INTERRUPT FROM IDLE DEVICE
160 ;               USER SUPPLIED ERROR RECOVERY ROUTINE
161 ;*****
162 PIER1:
163         JMP     PRTN    ; RETURN
164
```

```
165
166 ;*****
167 ;
168 ;       PRINTER OUTPUT DATA ROUTINE
169 ;
170 ;       CONTROL BLOCK ADDRESS IN D AND E REG
171 ;
172 ;*****
173 PDATA:
174         IN      PORTC   ; GET STATUS OF DEVICE
175         ANI     LPBSY   ; SEE IF BUSY (BUFFER FULL)
176         JZ      PD10    ; IF BUSY - BRANCH
177         LXI     H,CBCT  ; GET INDEX TO CT
178         DAD     D       ; COMPUTER ADDRESS OF CT
179         MOV     A,M     ; GET CT
180         INR     M       ; INC CT
181         DCX     H       ; DEC TO CC
182         CMP     M       ; SEE IF EQUAL
183         JZ      PCOMP   ; IF EQUAL - DONE GO TELL USER
184         LXI     H,CBUF  ; GET INDEX TO BUFFER ADDRESS
185         DAD     D       ; COMPUTE ADDRESS OF BUFFER ADDRESS
186         PUSH    D       ; SAVE D AND E REGISTERS
187         MOV     E,M     ; GET LSB OF BUFFER ADDRESS
188         INX     H       ; INC TO NEXT BYTE
189         MOV     D,M     ; GET BUFFER MSB
190         MVI     H,00H   ; CLEAR H REG
191         MOV     L,A     ; GET CT
192         DAD     D       ; COMPUTER CHARACTER ADDRESS
193         MOV     A,M     ; GET CHARACTER
194         OUT     PORTA   ; OUTPUT CHARACTER TO PRINTER
195         MVI     A,STBOF ; RESET DATA STROBE (LOW TRUE SIGNAL)
196         OUT     CWR
197         INR     A       ; GENERATE SET CONTROL WORD
198         OUT     CWR     ; SET DATA STROBE
199         POP     D       ; RESTORE CONTROL BLOCK ADDRESS
200         JMP     PDATA   ; LOOP UNTIL BUSY
201
```

The printer output data routine places a character in the output buffer of the 8255. Data in the control block is used to direct the transfer of a character. A data strobe signal is then generated through the use of the port C bit set/reset feature.

The flowchart and listing for the printer output data routine are shown below.



If the printer is busy at the time the printer output routine is called, a printer busy routine is executed. The printer busy routine disables the processor interrupts, enables the 8255 interrupts and then enables the processor interrupts on its return to the caller.

```
202
203 ;*****
204 ;       PRINTER BUSY - RETURN
205 ;*****
206 PD10:
207         DI              ; DISABLE INTERRUPTS
208         MVI     A,IEN   ; ENABLE DEVICE INTERRUPTS
209         OUT     CWR     ; SET INTERRUPT ENABLE
210         RET             ; RETURN TO CALLER
```

When the printer output routine has exhausted the data from the buffer, a good status code is posted to the user. The command in progress flag is also cleared.

```
211 ;*****
212 ;       POST GOOD COMPLETION TO USER
213 ;*****
214 PCOMP:
215         MVI     A,STGD  ; GET GOOD STATUS CODE
216         CALL    POST    ; POST TO USER
217         XRA     A       ; CLEAR A REG
218         STA     PIPRG+1 ; CLEAR COMMAND IN PROGRESS ADDRESS
219         RET             ; RETURN TO CALLER
220
```

The post to user completion routine obtains the completion address from the device control block and passes control to the user routine.

```
221
222 ;*****
223 ;
224 ;       POST TO USER COMPLETION ROUTINE
225 ;
226 ;               INPUTS:  STATUS CODE IN A REG
227 ;                        CONTROL BLOCK ADDRESS IN D AND E REG
228 ;               OUTPUTS: PASSES CONTROL TO USER COMPLETION ADDRES
229 ;                        SPECIFIED IN CONTROL BLOCK
230 ;                        WITH CONTROL BLOCK ADDRESS IN D AND E RE
231 ;
232 ;       A,H,L,B,C REG MODIFIED
233 ;
234 ;*****
```

```
235 POST:
236         XCHG
237         MOV     M,A.    ; UPDATE STATUS
238         XCHG
239         LXI     H,CBCMP ; GET INDEX TO COMPLETION ADDRESS
240         DAD     D       ; COMPUTE ADDRESS
241         MOV     C,M     ; GET LSB OF COMPLETION ADDRESS
242         INX     H       ; INC TO NEXT BYTE
243         MOV     B,M     ; GET MSB OF COMPLETION ADDRESS
244         PUSH    B       ; PUSH ADDRESS ON STACK
245         RET             ; PASS CONTROL TO USER ROUTINE
246 ;*****
247 ;       DATA AND TABLES
248 ;*****
249         ORG     3D00H
250 PIPRG:  DW      0       ; IN PROGRESS CONTROL BLOCK ADDRESS
251                         ; IF DATA = 0 NO CONTROL BLOCK IN PROGRESS
252                         ; IF DATA <> 0 CONTROL BLOCK IN PROGRESS
253 ;*****
254 ;       END OF MODE ONE EXAMPLE
255 ;*****
256         END
```

SUMMARY/CONCLUSIONS

The iSBC 80/10A has the capability to function in the capacity of a peripheral processor and/or a device controller. This capability is provided in part by the interrupt support logic accompanying the parallel I/O ports. This application example shows how the iSBC 80/10A requires only an interconnect to the device to be controlled.

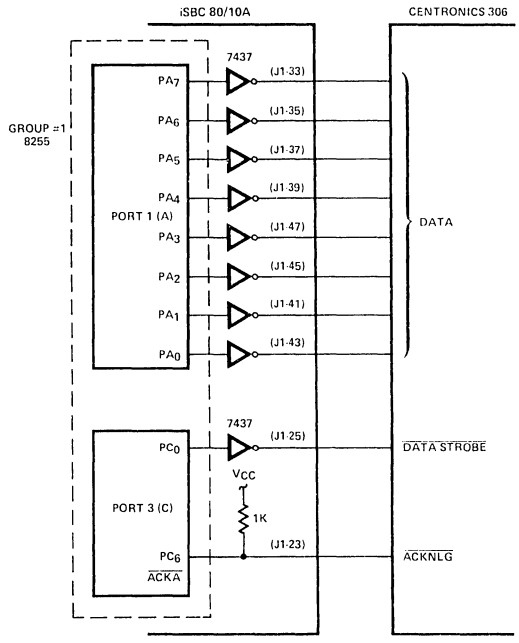


Figure 11. Printer Controller Schematic

CONCLUSION

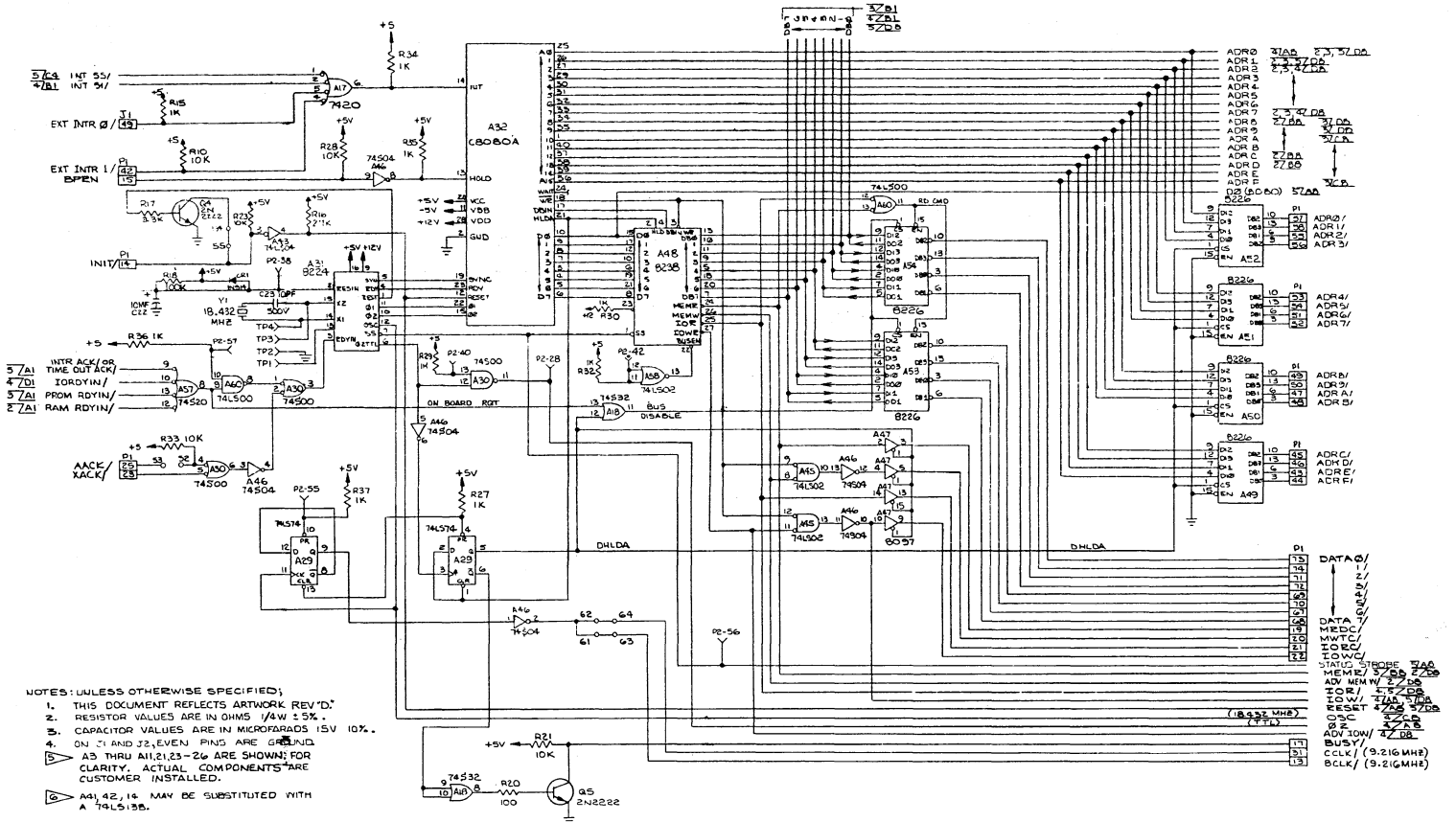
The purpose of this application note has been to expose the reader to a broad spectrum of potential applications of the Intel iSBC 80/10A and System 80/10 products. Applications have been presented in the areas of instrumentation, communication, process control and I/O device control. The examples were limited to short problems that could be completely described.

Intel's PL/M-80 and 8080 Macro Assembly Language were both used in the examples. Instead of using only assembly language, it was felt that PL/M-80 should also be shown. Coding in an algorithmic language is generally more natural than assembly language and provides these added benefits: reduced program development time and cost, improved product reliability, and easier program maintenance.

While the task of actually configuring the SBC 80/10 for the applications has not been described in this note, detailed instructions are contained in the tables of Chapter 4 in the*iSBC 80/10 and iSBC 80/10A Single Board Computer Hardware Reference Manual.*

The Intel iSBC 80/10A has been designed to provide users with subsystems that have processing power, memory storage, parallel and serial programmable I/O interfaces. A design goal of the iSBC 80/10A was to minimize the requirements for customized interface hardware in user applications. This application note has demonstrated the achievement of this goal. The net effect is to reduce the number of tedious design tasks, thus allowing the systems designer to concentrate on systems integration and other problems unique to his job.

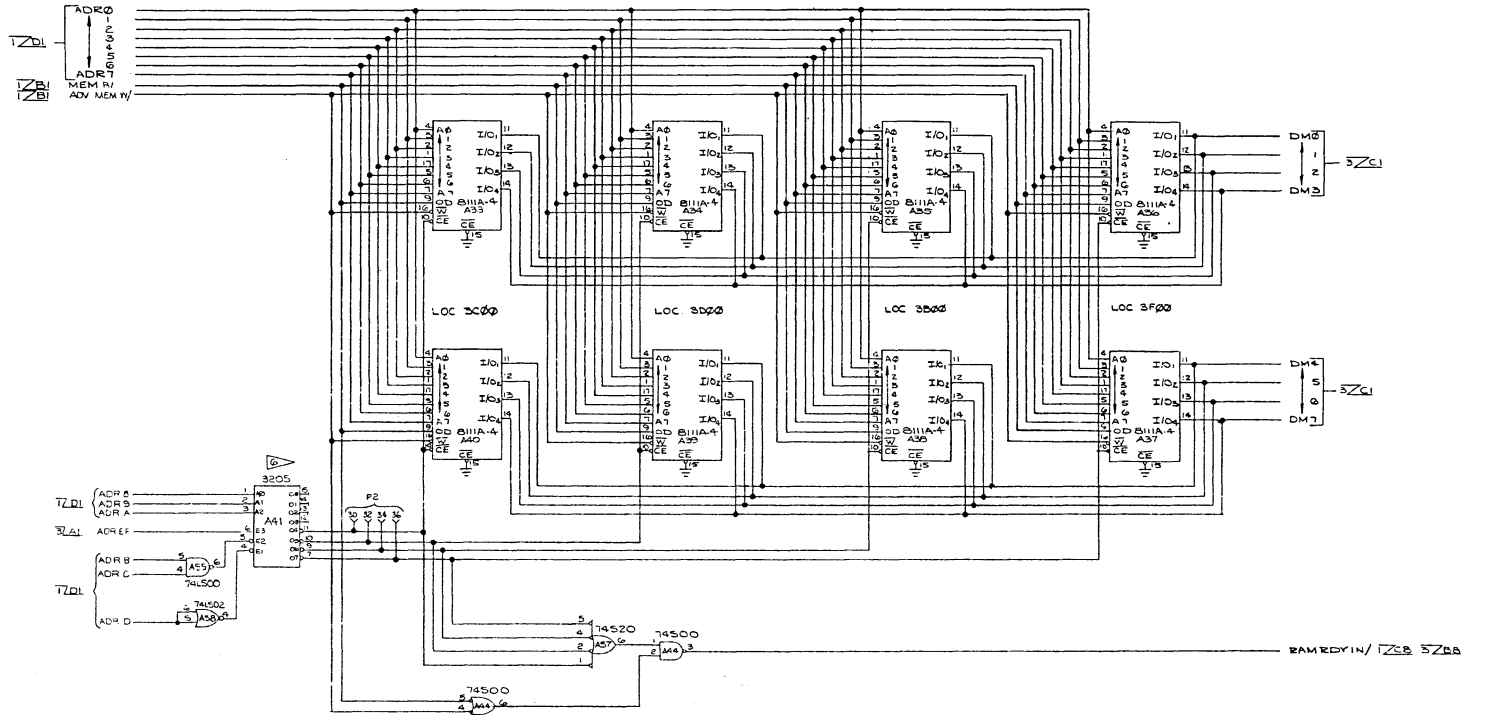
APPENDIX A
iSBC 80/10A SCHEMATICS



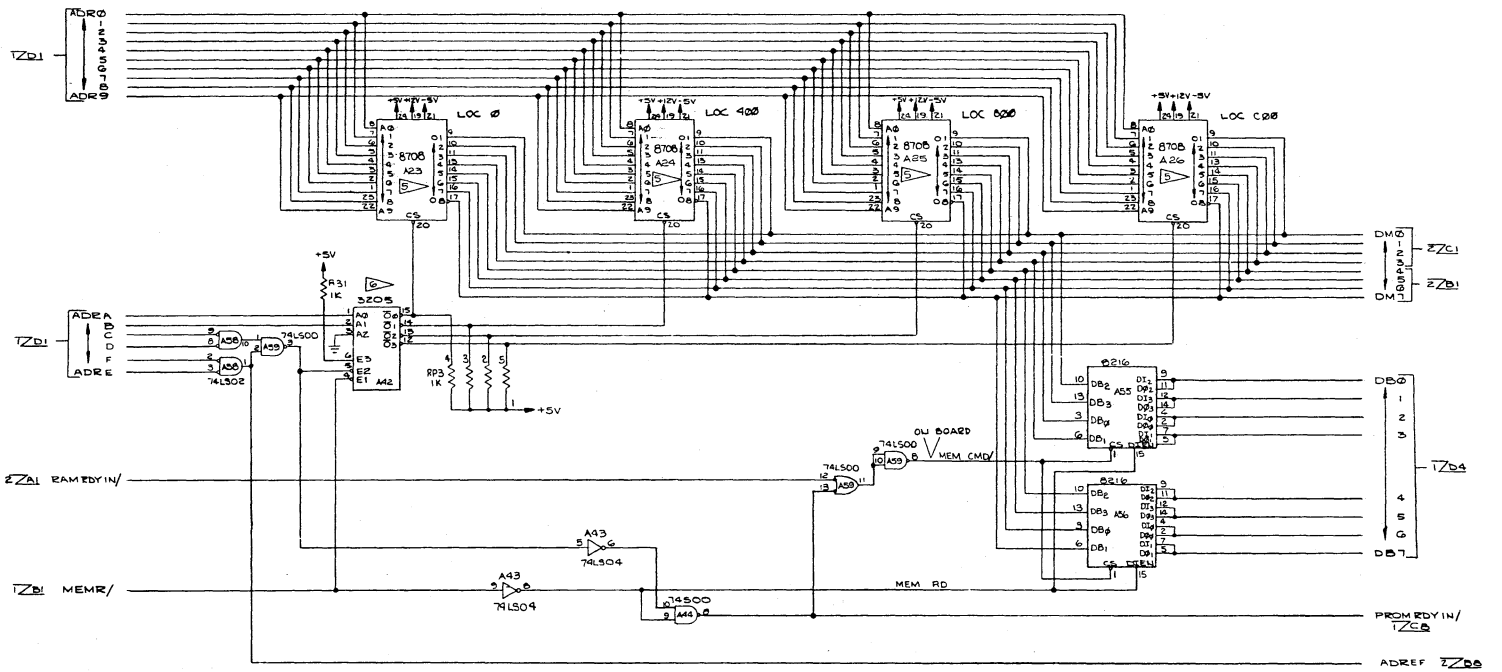
NOTES: UNLESS OTHERWISE SPECIFIED;

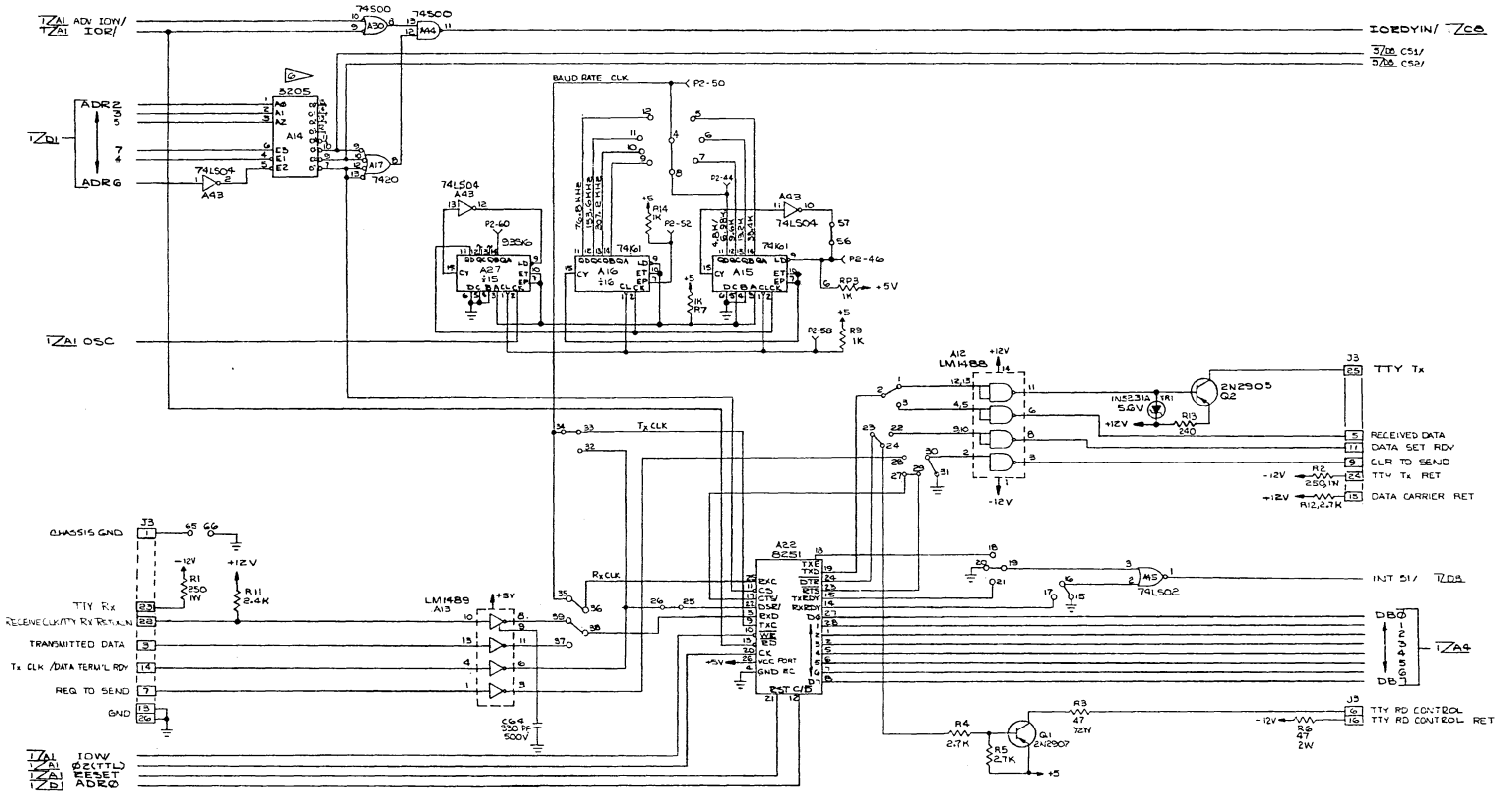
1. THIS DOCUMENT REFLECTS ARTWORK REV 'D.'
2. RESISTOR VALUES ARE IN OHMS 1/4W ± 5%.
3. CAPACITOR VALUES ARE IN MICROFARADS 15V 10%.
4. ON J1 AND J2, EVEN PINS ARE GROUND.
5. A3 THRU A11, 21, 23-26 ARE SHOWN; FOR CLARITY. ACTUAL COMPONENTS ARE CUSTOMER INSTALLED.
6. A41, 42, 14 MAY BE SUBSTITUTED WITH A 74LS138.

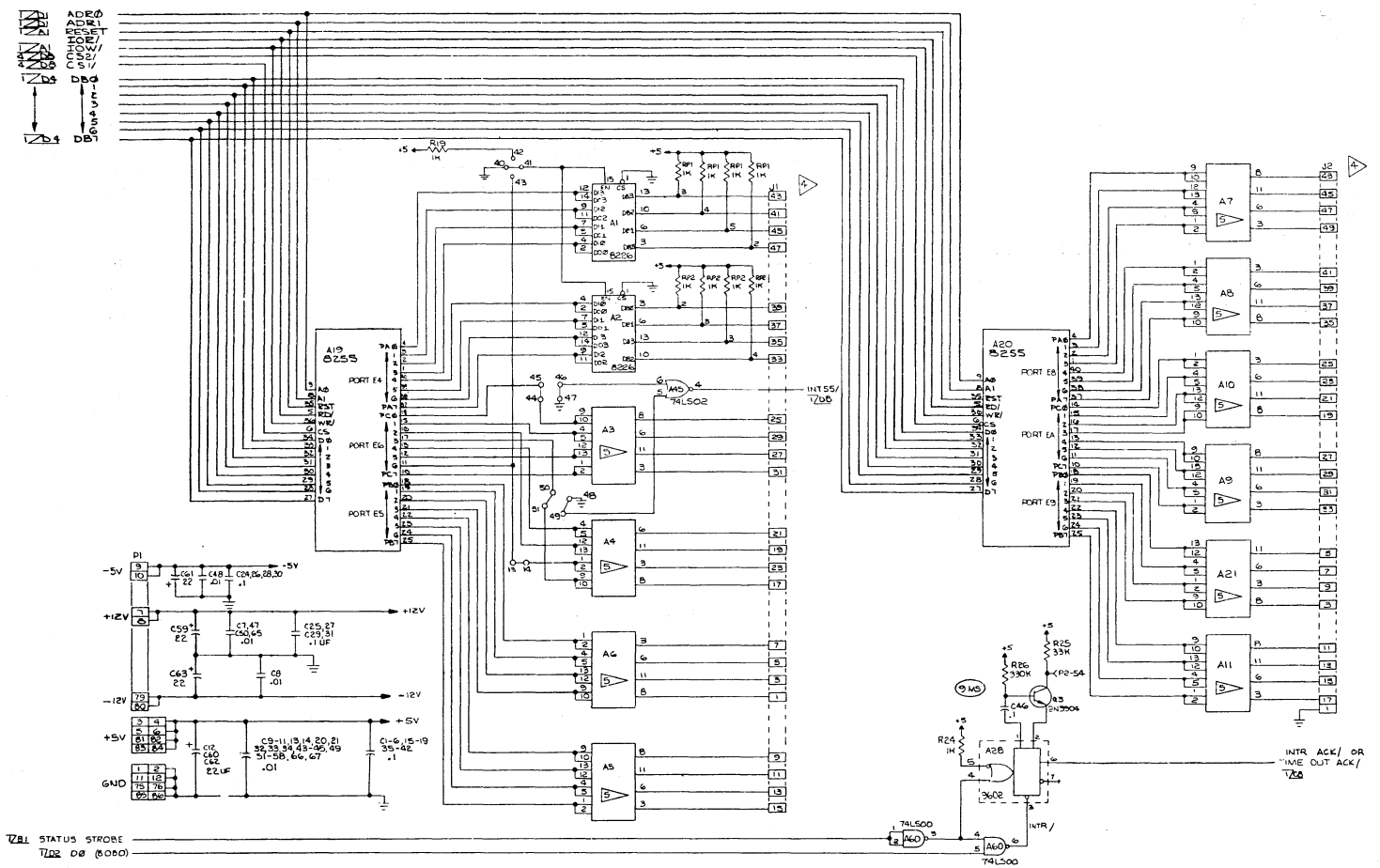
INFORMATION AND SCHEMATICS SUBJECT TO CHANGE WITHOUT NOTICE, FOR REFERENCE ONLY.

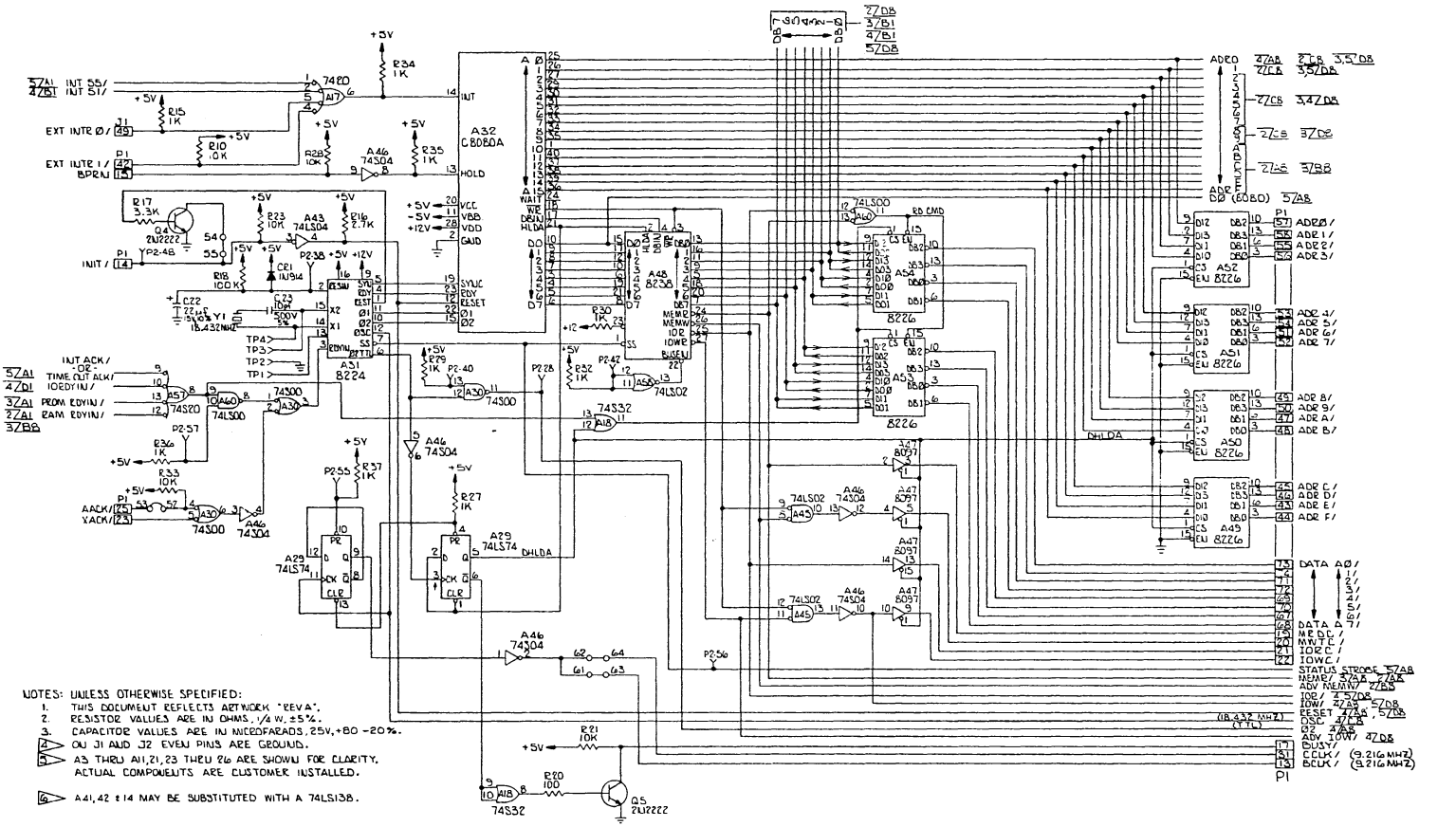


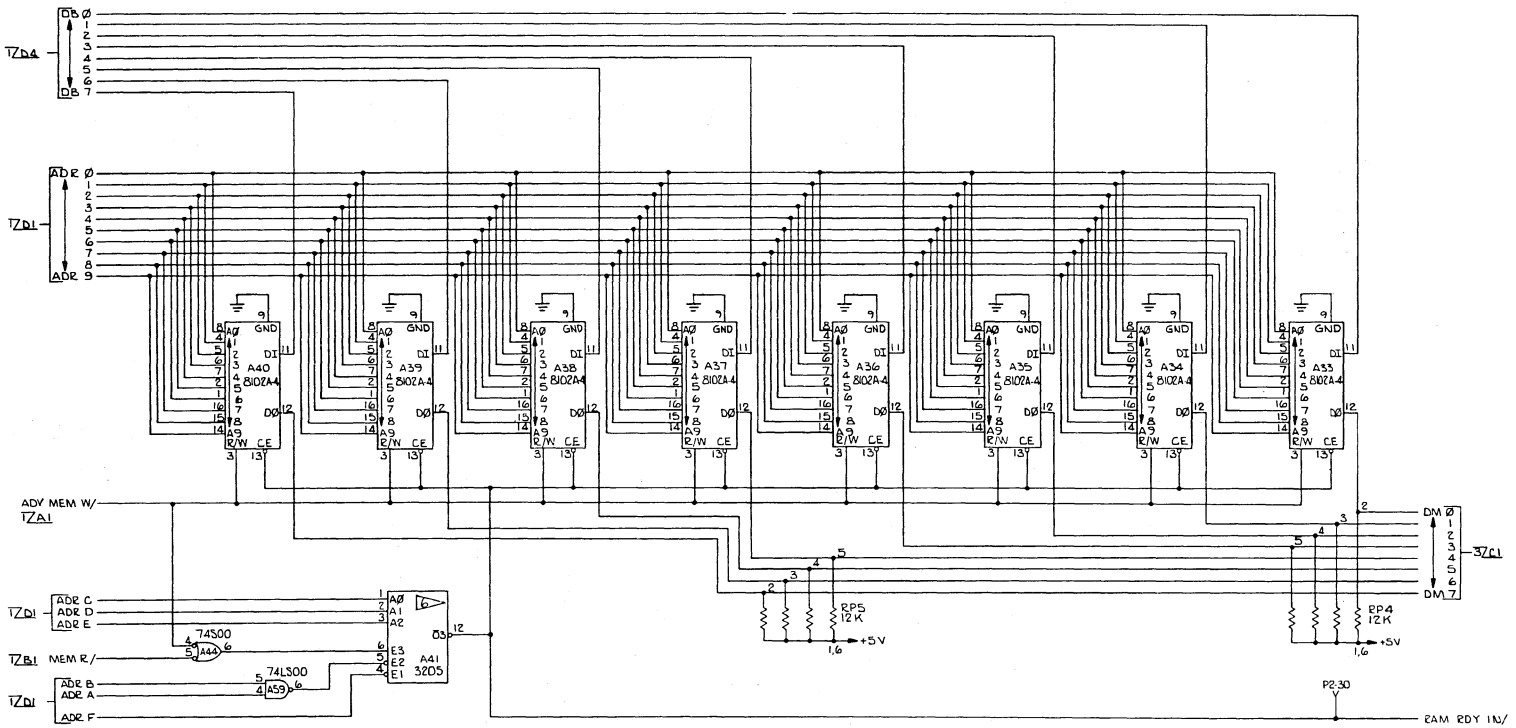
INFORMATION AND SCHEMATICS SUBJECT TO CHANGE WITHOUT NOTICE, FOR REFERENCE ONLY.

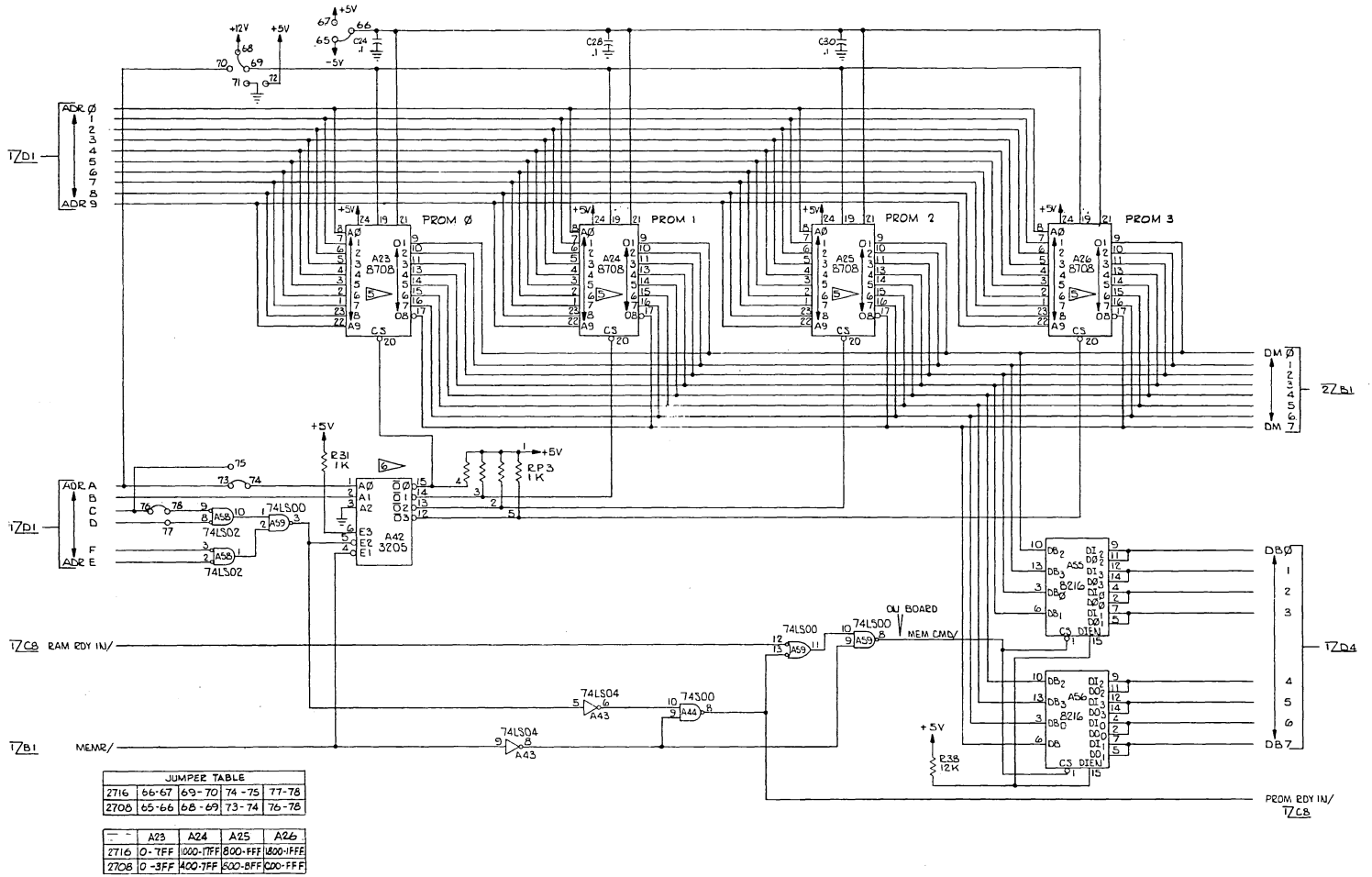







NOTES: UNLESS OTHERWISE SPECIFIED:
1. THIS DOCUMENT REFLECTS ARTWORK "REVA".
2. RESISTOR VALUES ARE IN OHMS, 1/4 W. ±5%.
3. CAPACITOR VALUES ARE IN MICROFARADS, 25V, +80 -20%.
4 ON J1 AND J2 EVEN PINS ARE GROUND.
5 A3 THRU A11, 21, 23 THRU 26 ARE SHOWN FOR CLARITY. ACTUAL COMPONENTS ARE CUSTOMER INSTALLED.
6 A41, 42 & 14 MAY BE SUBSTITUTED WITH A 74LS138.
INFORMATION AND SCHEMATICS SUBJECT TO CHANGE WITHOUT NOTICE, FOR REFERENCE ONLY.

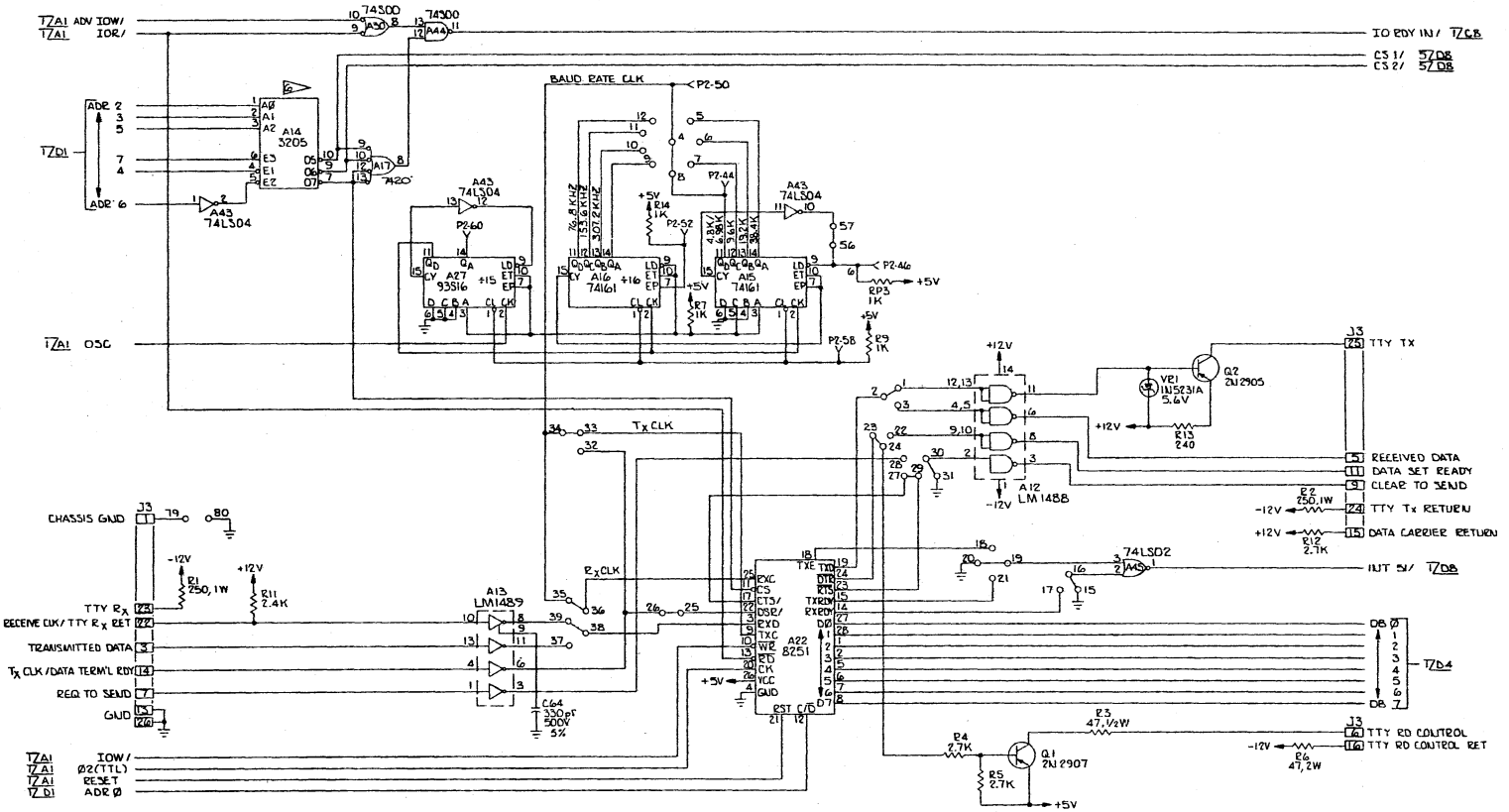




JUMPER TABLE				
2716	66-67	69-70	74-75	77-78
2708	65-66	68-69	73-74	76-78

—	A23	A24	A25	A26
2716	0-7FF	1000-17FF	1800-1FFF	
2708	0-3FF	400-7FF	800-BFF	C00-FFF

INFORMATION AND SCHEMATICS SUBJECT TO CHANGE WITHOUT NOTICE, FOR REFERENCE ONLY.



INFORMATION AND SCHEMATICS SUBJECT TO CHANGE WITHOUT NOTICE, FOR REFERENCE ONLY.