Spring 1979

An Application of a Single Chip Slave Microcomputer as an Intelligent Interface

Thomas J. Riordan
University of Central Florida

AN APPLICATION OF A SINGLE CHIP SLAVE MICROCOMPUTER AS AN INTELLIGENT INTERFACE

BY

THOMAS JAMES RIORDAN
B.S.E., Florida Technological University, 1978

THESIS

Submitted in partial fulfillment of the
requirements for the degree of Master of Science in
Engineering in the Graduate Studies Program of the
College of Engineering of University of Central Florida
at Orlando, Florida

Spring Quarter
1979

ABSTRACT

This thesis describes the operation of an intelligent controller existing between a computer system and a peripheral. The intelligent controller is implemented with an INTEL universal peripheral interface, UPI-41, single chip slave microcomputer. The interfaces between the controller and the computer system, and the controller and the peripheral are described in detail, as are the firmware and internal facilities of the controller itself. An evaluation of system operation is presented and, finally, the organizational philosophy of the system is discussed. Located within the appendicies are a system schematic and listing of the controller program.

TO: DR. HERBERT C. TOWLE

ACKNOWLEDGEMENT

The author wishes to thank Mr. Albert Marshal at the Orlando Naval Training Center for his involvement in the preparation of this thesis. A special thanks goes to Mrs. Doug Caldes for her efforts in typing the final draft.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

This thesis describes the hardware and software of an intelligent controller system. The controller regulates communication between a microcomputer system, an Intel SBC 80/20 (1), a CRT display, the Lear Seigler ADM-3A (2,3), and a set of control switches. Controller components include:

1. An Intel Universal Programmable Interface, UPI-41, single chip microcomputer (4)
2. A 7474 Dual D Flip Flop (5)
3. A 9602 Dual One Shot (6)
4. And a 75188 RS-232C Inverting Line Driver (7)

The system schematic and the UPI-41 control program are located in appendices 1 and 2 respectively.

The system description is divided into four sections: in section II, a functional summary and a component interface description are presented. Performance criteria are also established in this section. Section III describes the facilities available within the UPI-41 and explains their use in the present application. Section IV describes the UPI-41 control program and section V evaluates the system with respect to assumption validity, performance criteria, and maximum system capabilities. Finally, in the conclusion a discussion of the general organizational philosophy of the system is presented.

II. SYSTEM DESCRIPTION

The function of the intelligent controller is to receive parallel data from the SBC 80/20, decode the data, and cause a message to appear on the ADM-3A screen based upon the content of the data received. The control switch settings also affect controller operation, but only secondarily.

A block diagram showing the system component relationships appears in figure 1.

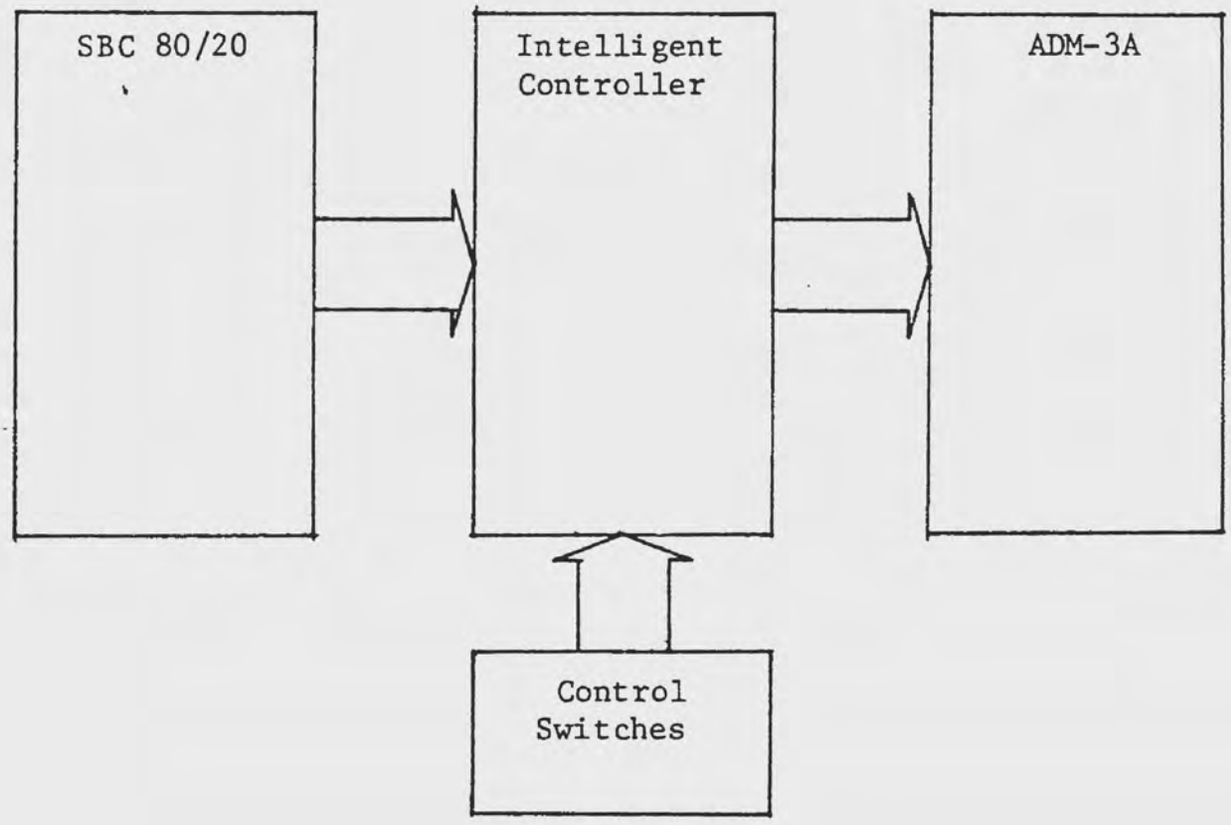


Fig. 1. System Block Diagram

The following describes the three component interfaces shown in figure 1: SBC 80/20 to controller, controller to ADM-3A, and control switches to controller.

SBC 80/20 To Controller Interface

The SBC 80/20 to controller interface is comprised of three sets of connections. The first set, consisting of 8 data lines and 1 control line, are the data transfer connections. The second set consists of the clock connections, while the third set consists of only one connection, the initialization connection.

Data Transfer Connections

The 8 data lines of the data transfer set connect an 8 bit output port on the SBC 80/20 to the 8 bit Interface Register of the UPI-41. There are six I/O ports on the SBC 80/20 numbered 1 through 6 (1). These ports are divided into the Group A ports, 1 - 3, and the Group B ports, 4 - 6. Each port group corresponds to a single 8255 Programmable Peripheral Interface, PPI (8). Port 4 of Group B is programmed as an output port and used for the SBC 80/20 to UPI-41 data connection.

To transmit data to the UPI-41, the SBC 80/20 places data on port 4 and sends a Data-Available pulse to the UPI-41 over the control line. The Data Available pulse is software generated and is transmitted through port 3 of the Group A 8255. The length of the Data Available pulse is set by the time required to execute the instructions necessary to change the logic level of the control line

twice, first from high to low, then from low to high. For the SBC 80/20 this results in a 10 microsecond pulse. The maximum pulse length to the UPI-41 is set at twice the instruction cycle length, or 6.5 microseconds (4); therefore, the 10 microsecond Data Available pulse is sent to the one shot within the controller where it is shortened to 1 microsecond. The 1 microsecond pulse from the one shot supplies the $\overline{WR}$ input to the UPI-41. On the rising edge of this pulse the data on the SBC 80/20 output port is latched into the UPI-41 Interface Register. SBC 80/20 to controller data transfer connections are illustrated in figure 2.

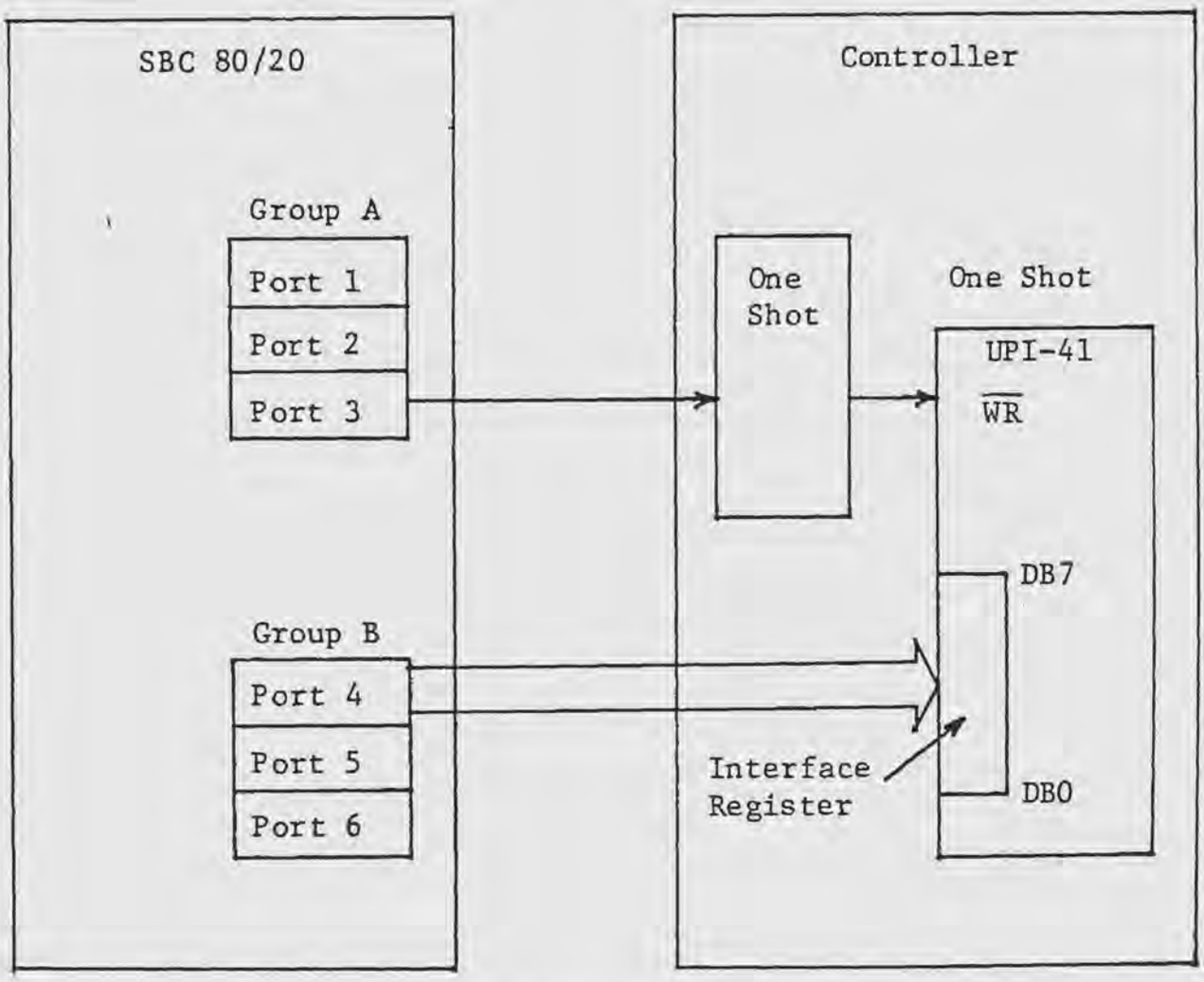


Fig. 2. SBC 80/20 to controller

Each byte of data transferred from the SBC 80/20 to the UPI-41 contains two kinds of information encoded into separate fields within the byte. The three most significant bits contain a source identifier encoded in straight binary, and the four least significant bits contain a message identifier, also in straight binary, see figure 3. Bit four is not used.

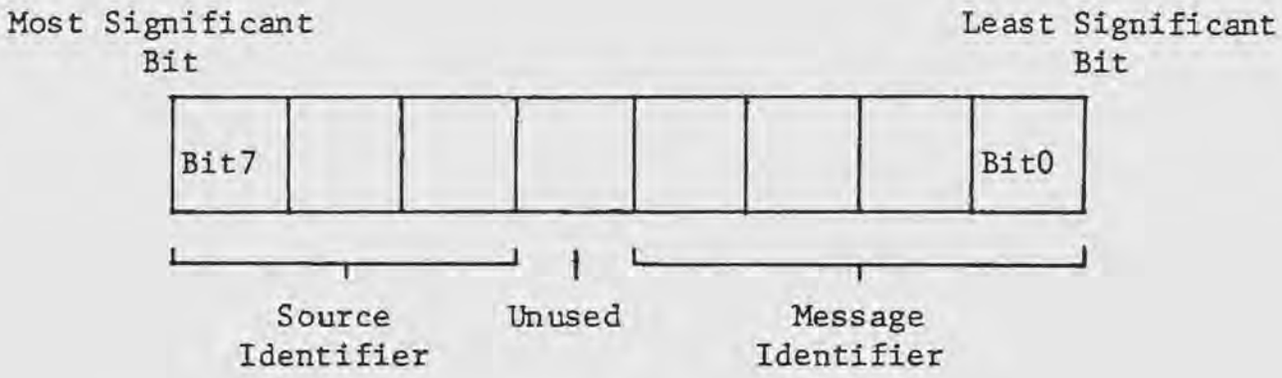


Fig. 3. SBC 80/20 to UPI-41 data byte

The rate of data transfer from the SBC 80/20 to the controller can be characterized by three separate data transfer rates of which the last two will be of interest. The first two rates are determined by the SBC 80/20 input configuration, figure 4, while the third is determined by the input configuration in combination with the SBC 80/20 data processing rate.

The SBC 80/20 input configuration consists of 5 input sources, where each source contains a data latch and a service request line. When data is latched into one of the sources, the SBC 80/20 receives a service request signal from that source. For each

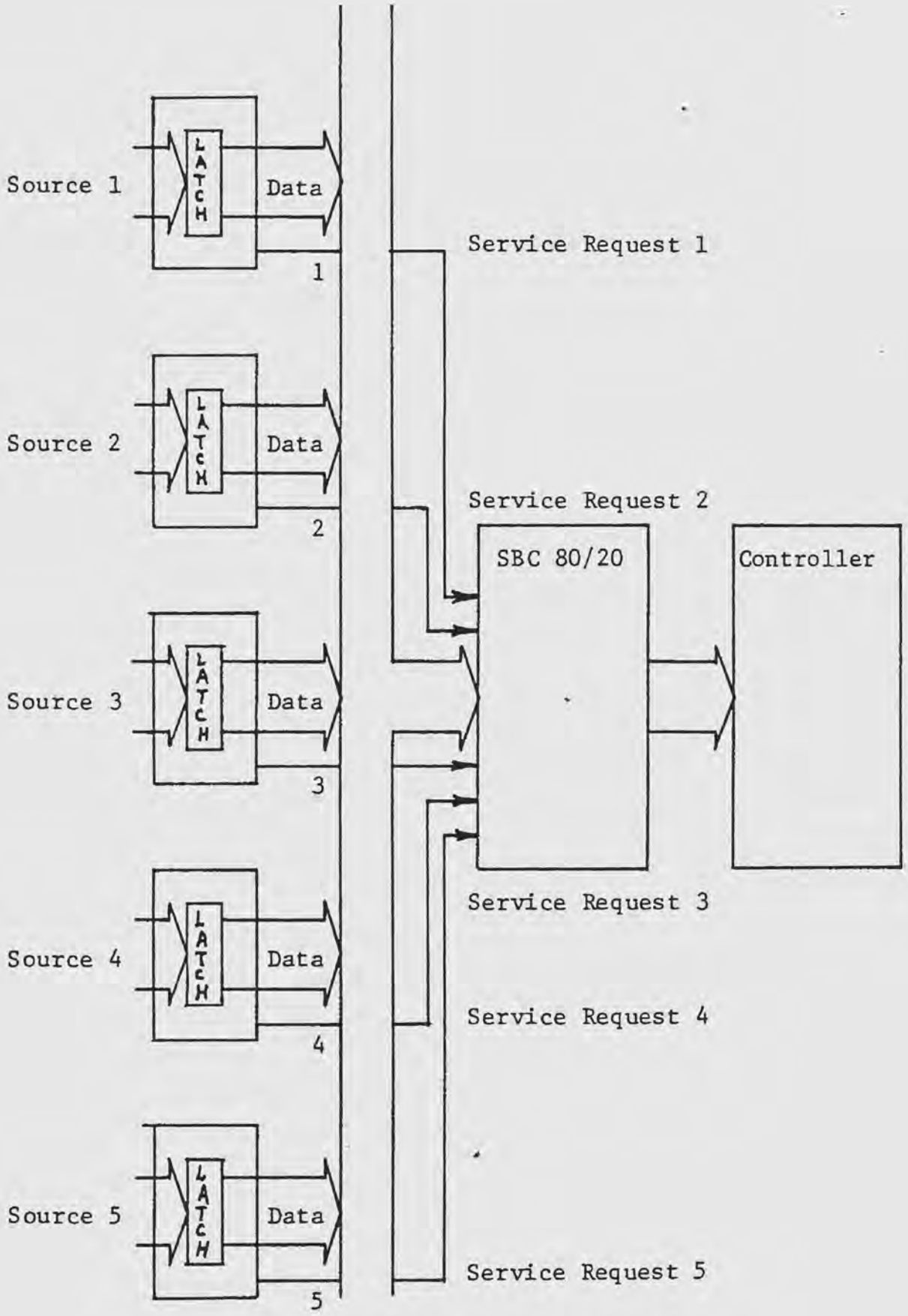


Fig. 4. SBC 80/20 input source configuration

service request that the SBC 80/20 responds to, a data byte will be sent to the controller.

The first data transfer rate is the average transfer rate and occurs when the 5 sources are initiating service requests at their nominal rate. The second data transfer rate is a peak average rate, and occurs when all 5 sources are initiating service requests at their maximum rate of 12 per second. This condition results in a peak average rate of 12 x 5, or 60 transfers per second. The third data transfer rate is the maximum rate, and occurs anytime there are simultaneous service requests to the SBC 80/20. This rate is determined by the processing rate of the SBC 80/20. Analysis using : (1) real time emulation under control of Intel's In Circuit Emulator, ICE-80 (9,10), (2) tabulation of instructions executed and their execution time (11), and (3) experimental determination (11), indicates that the SBC 80/20 processing rate is approximately 200 inputs per second.

As indicated before, the peak average transfer rate of 60 transfers per second, and the maximum transfer rate of 200 transfers per second are the relevant quantities characterizing the data transfer interface.

To keep up with the SBC 80/20 over extended periods, the processing rate of the UPI-41 must equal or exceed the SBC 80/20 peak average transfer rate, and to keep up with the SBC 80/20 when simultaneous service requests have occured, the reception rate of the UPI-41 must equal or exceed the SBC 80/20 maximum transfer rate.

The requirement on the UPI-41 processing rate will be used in the sequel to determine the baud rate used in the controller to ADM-3A interface, while the requirement on the UPI-41 reception rate will be used to establish the necessity of a data queue within the UPI-41.

One final point is that there are no provisions for the UPI-41 to indicate that it is ready to accept a data transfer from the SBC 80/20. Thus, the data queue mentioned above will be filled by an interrupt driven procedure. This technique will assure that a data byte has been removed from the Interface Register before an additional data transfer can occur.

Clock Connections

The clock connections supply the UPI-41 clock inputs, X1 and X2. A single line from the SBC 80/20 supplies the controller with a 9.216 megahertz clock which the SBC 80/20 makes available as the BCLK output. Within the controller, the BCLK frequency is divided in half by a D flip flop within the 7474. This division is necessary to bring the BCLK frequency within the 1 to 6 megahertz operating range of the UPI-41 (4). The Q and $\overline{Q}$ outputs of this flip flop supply the UPI-41 inputs, X1 and X2, with a 180^{o} out of phase 4.608 megahertz clock. While the UPI-41 is capable of generating its own clock by connecting a crystal to the X1 and X2 inputs, the BCLK frequency is used since the standard asynchronous communication frequencies can be derived from it. The clock connections are shown in figure 5.

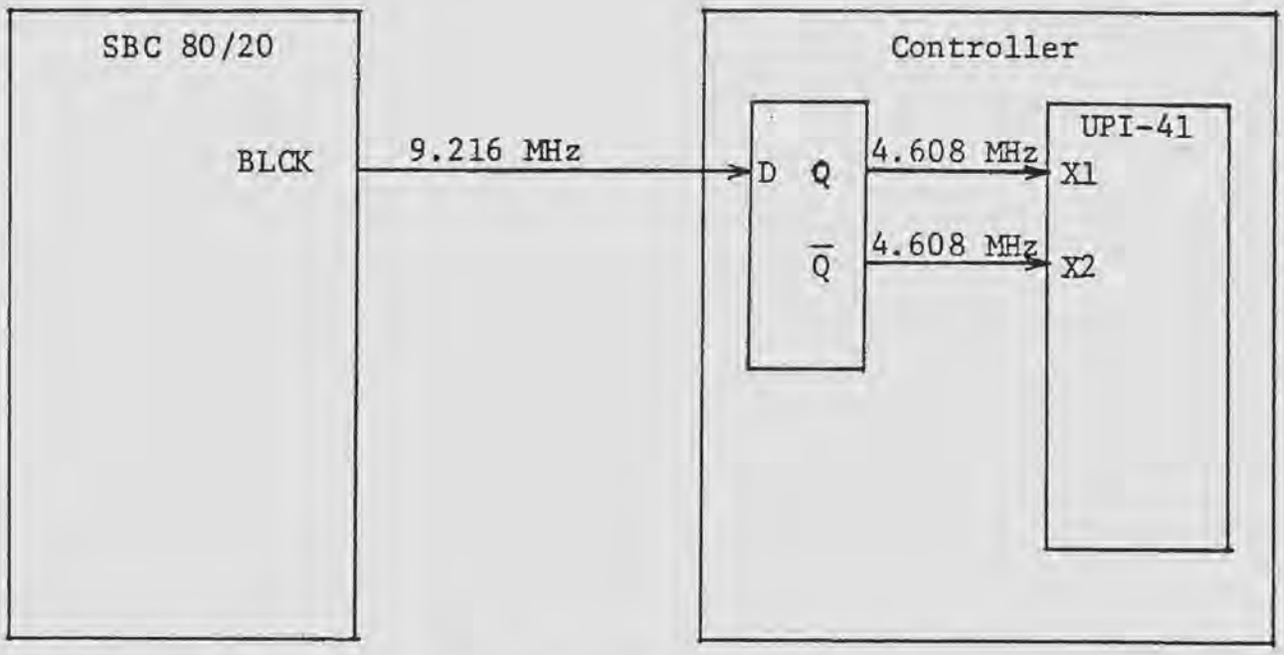


Fig. 5. SBC 80/20 to controller - clock connections

Initialization Connection

The initilization connection is between the $\overline{\text{INIT}}$ output of the SBC 80/20 and the $\overline{\text{RESET}}$ input of the UPI-41. A low going pulse on this line causes the control program of the UPI-41 to begin execution at location 0.

Controller to ADM-3A Interface

The controller to ADM-3A interface consists of a single line which originates from line 0 of port 1 on the UPI-41, passes through the 75188 inverting line driver, and terminates on the Receive Data, RXD, input of the ADM-3A. The line driver converts the TTL output of port 1, 0 - 5 volts, into RS-232C logic levels of $\pm$ 12 volts.

Information is transmitted from the UPI-41 to the ADM-3A serially using 7 bit ASCII code under the RS-232C communication

protocol (2). For this application, the number of bits per character has been minimized by using a single stop bit and no parity bit. For a given serial transmission rate this configuration will result in the fastest possible character transmission time. This time is an important consideration, as each parallel byte received by the controller from the SBC 80/20 will require a 22 character message to be transmitted. With the single start bit, the 9 bit serial character appears as shown in figure 6.

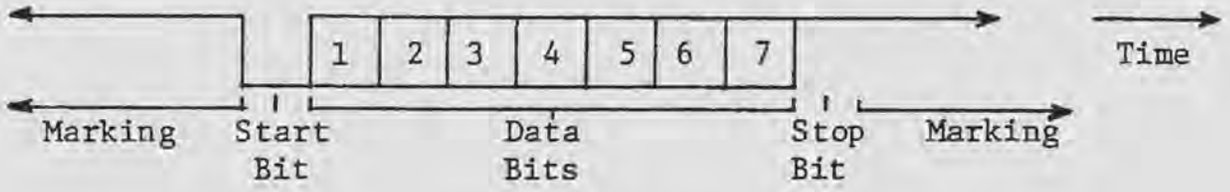


Fig. 6. Serial transmission character

Each data byte received by the UPI-41, except as noted in the next section, causes a string of 9 bit characters to be sent from the UPI-41 to the ADM-3A, a 24 line by 80 character CRT display.

The function of the ADM-3A is to provide three kinds of information concerning the SBC 80/20 inputs to an observer. The ADM-3A displays a message, indicates the SBC 80/20 source corresponding to the message, and reflects the order of input occurrence. The message is indicated by the characters displayed on the screen. The source is indicated by dividing the ADM-3A screen into 5 columns of equal width, with the first column reserved for source 1 messages, the second column for source 2 messages, and so on for the five

sources. The order of inputs is indicated by scrolling the display 1 line each time a message is displayed.

For a screen width of 80 characters, and not allowing an overlap of columns, the message field for each source is limited to the integer portion of 80/5, or 16 characters. The ADM-3A screen use is illustrated in figure 7.

To implement the function of the ADM-3A as described above requires that 22 characters be sent to the ADM-3A for each SBC 80/20 to controller transfer. The 22 characters are sent in 3 groups: a cursor control group, a message group, and a display control group.

The first group sent, the cursor control group, contains four characters which cause the cursor of the ADM-3A to position itself at the beginning of one of the five message columns. The first two control characters "escape" and "equals", activate the ADM-3A cursor positioning logic, while the next two characters are interpreted as the X and Y coordinates of the new cursor position, respectively. The Y coordinate sent is always the same, 037H, and selects the bottom line of the display. The X coordinate is determined by the SBC 80/20 input source.

The second group sent, the message group, contains 16 characters. These characters will be printed on the screen of the ADM-3A in the message field whose beginning was established by the cursor positioning control group.

The third group sent, the display control group, contains the remaining 2 characters. These characters, a carriage return and line feed, cause the display to scroll up one line in prepara-

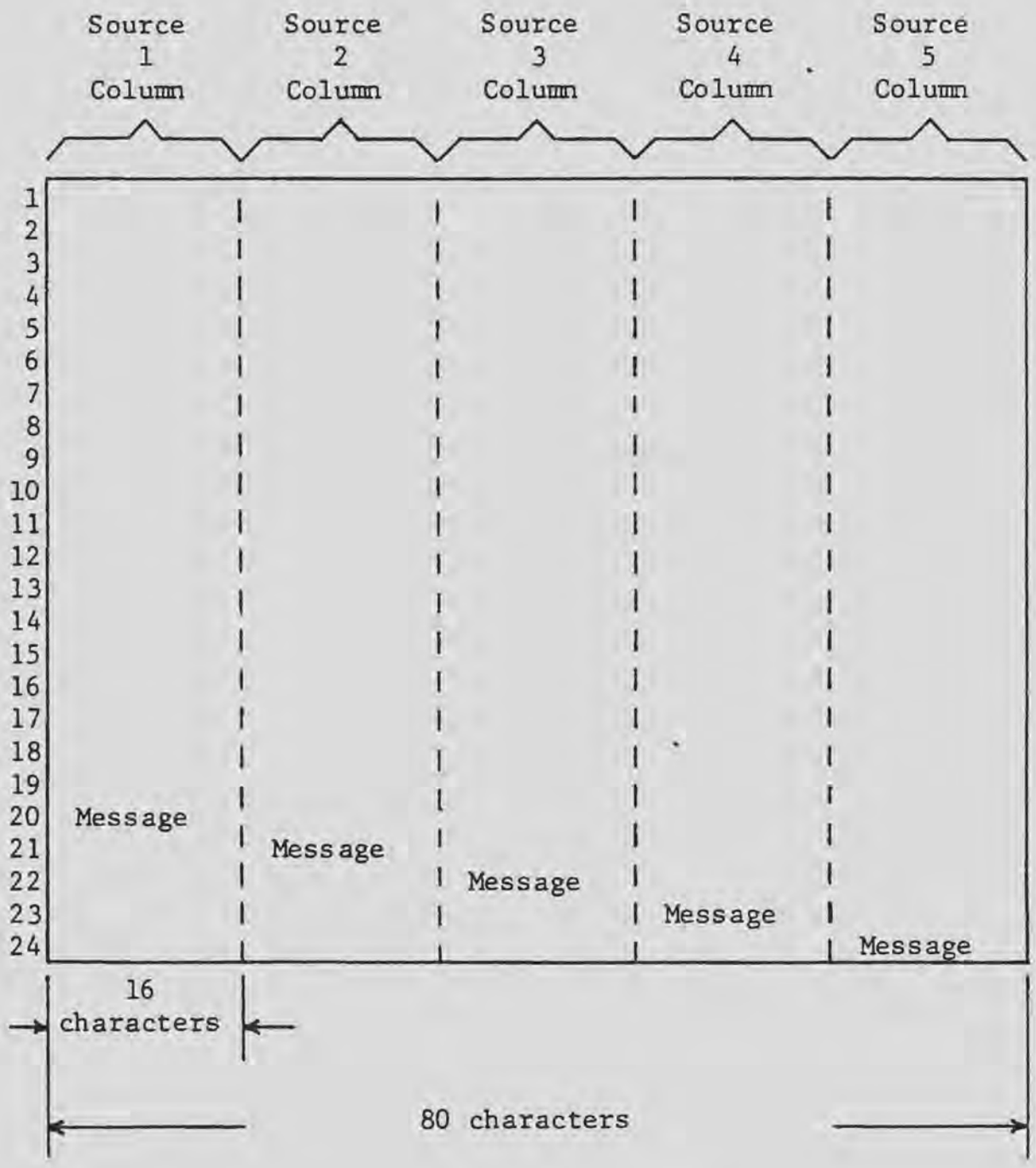


Fig. 7. ADM-3A screen use

tion for the next control group 1 sequence.

The complete 22 character string appears as shown in figure 8.

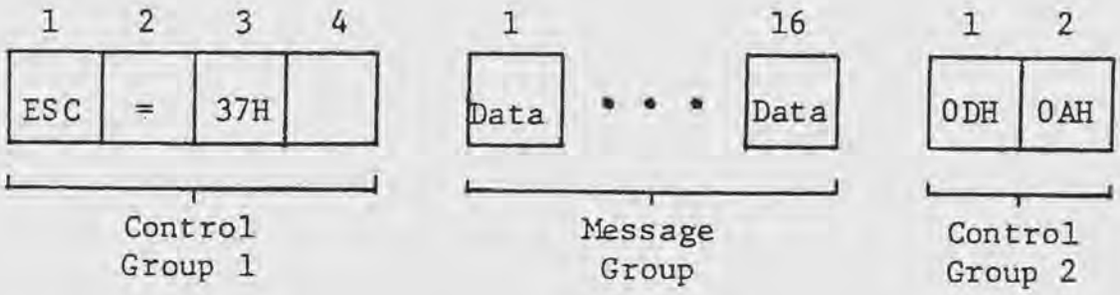


Fig. 8. Character string transmitted to ADM-3A

The final aspect of the controller to ADM-3A interface is the serial transmission rate to be used. Having now established (1) the number of characters sent by the controller to the ADM-3A per SBC 80/20 input, 22, the number of serial bits per character, 9, and (3) the UPI-41 processing rate requirement, 60 transfers/sec, a minimum serial transmission, or baud, rate can be computed as

$$\text{minimum baud rate} = \begin{array}{l} 22 \text{ characters/SBC 80/20 transfer } x \\ 9 \text{ bits/character } x \\ 60 \text{ SBC 80/20 transfers/second} \end{array} \qquad (1)$$

or 11,880 bits per second. The next highest, indeed the highest, baud rate at which the ADM-3A can receive data is 19,200 baud. This value must necessarily be chosen as the data transmission rate.

Control Switches to Controller Interface

The control switches to controller interface is a 5 line connection between 5 control switch outputs and the 5 least significant inputs of port 2 on the UPI-41. The design of port 2 on the UPI-41 is such that if nothing is connected to a port line, the line will read as a logic one, whereas, if the line is grounded through a 1k resistor, the port will read a logic zero (3). The control switch to controller connections are shown in figure 9.

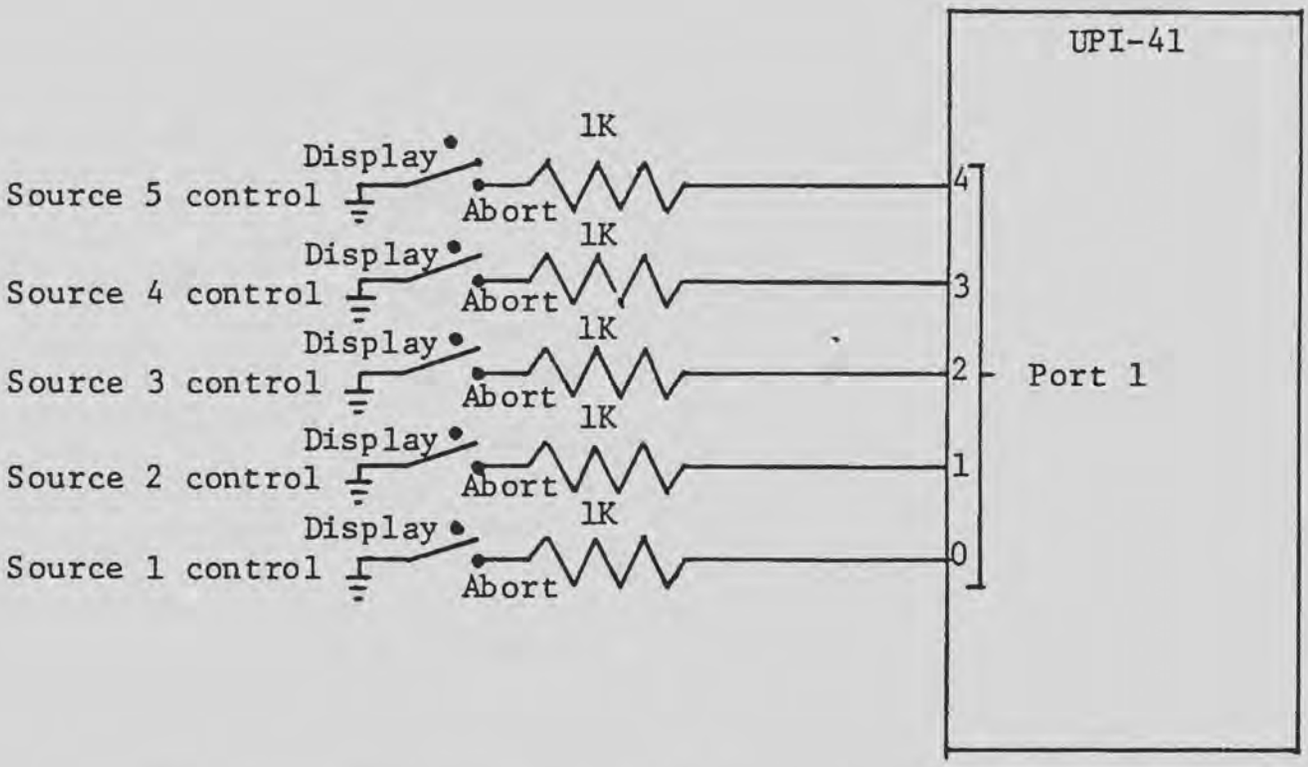


Fig. 9. Control switches to controller interface

During the processing of a data byte by the UPI-41, the binary source identifier is translated into a linear select code which is then compared with the switch settings on port 2. If the switch corresponding to the source identifier is set in the abort

position, a logic 0 is present and a message will not be sent. This is the exception referred to in the controller to ADM-3A interface description. If the switch is set in the display position a logic 1 will be present and a message will be sent.

This concludes the overall system description. The next two sections will describe the principal device within the intelligent controller, the UPI-41 single chip microcomputer.

III. UPI-41 SINGLE CHIP MICROCOMPUTER

The UPI-41 single chip microcomputer provides the intelligence of the intelligent controller. The block diagram below illustrates the facilities available within the UPI-41.

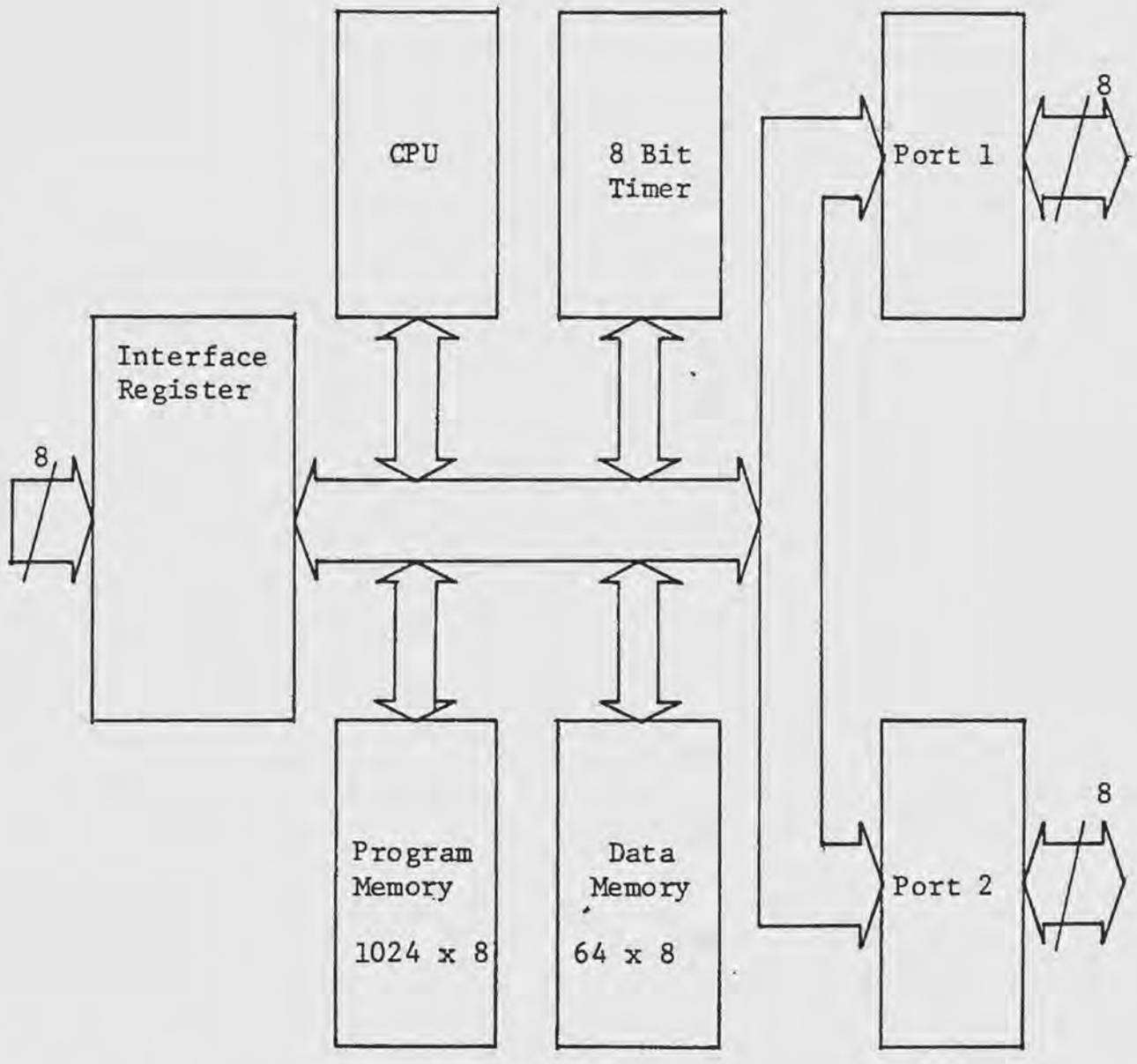


Fig. 10. UPI-41 single chip microcomputer block diagram

RAM within the UPI-41 serves three purposes: it contains the registers, the subroutine and interrupt stack, and the variable data storage locations. The distribution of the 64 RAM locations between these three functions is shown in figure 12.

The registers in bank 0 are designated R0-R7, while those in bank 1 are designated R0'-R7'. Only one register bank at a time can be addressed. Bank selection is accomplished by executing a special select register bank X, SEL RBX, instruction where X is either 0 or 1. The registers of bank 0 are used for data processing and message transmission, while those of bank 1 are used for queue control.

The UPI-41 contains a rather sophisticated timer which was evaluated for use as the bit interval generator for UPI-41 to ADM-3A serial transmission. As several difficulties were encountered, see appendix 3, the use of the timer, while representing a possible area for future research, was rejected in favor of a software timing approach. The software timing routine will be described, along with the rest of the UPI-41 control program, in the next section.

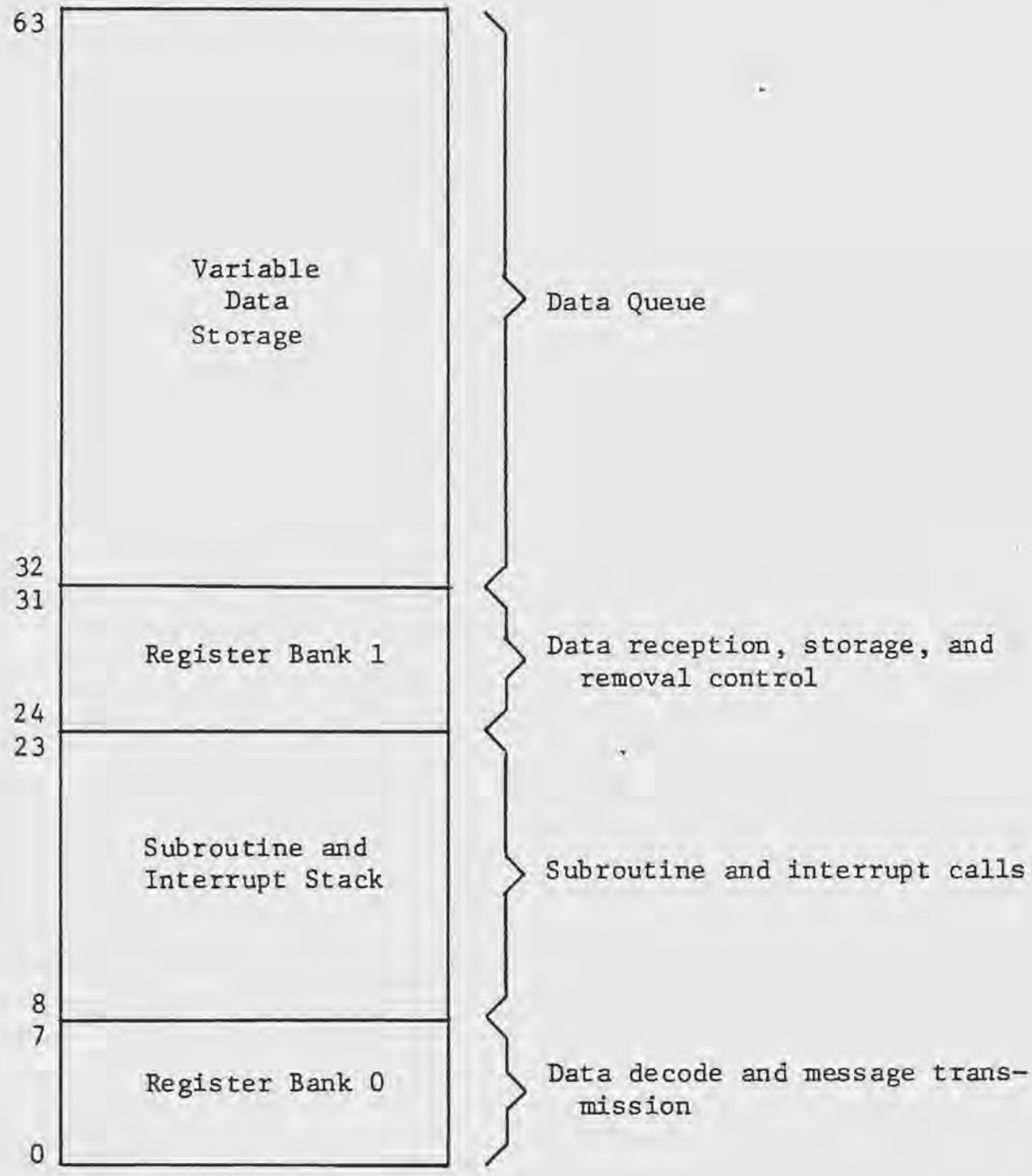


Fig. 12. UPI-41 RAM memory map

IV. UPI-41 CONTROL PROGRAM

The UPI-41 program is written in MCS-48/UPI-41 assembly language (12). The program was assembled using a cross assembler (13) operating on an Intel Microcomputer Development System, MDS-800 (14,15). The machine code was burned into the EPROM program memory of the UPI-41 (4) using an Intel Universal Prom Programmer (16) and the Universal Prom Mapper Software (17). The assembly of the program and the burning of the EPROM were done under control of the Intel System Implementation Supervisor, ISIS II (18), operating from an Intel Dual Floppy Disk Drive (19).

The program description is divided into three parts:

1. Initialization procedures
2. Data reception and storage
3. Data decode and message transmission

The program listing is located in appendix 2, a flow chart appears in figure 13a and 13b.

Initialization Procedures

The first section of the UPI-41 program performs functions which are necessary prior to data reception. These functions are the initialization of registers and the initialization of the ADM-3A screen. The values placed in the various registers will be explained as they are encountered within the program. The screen initializa-

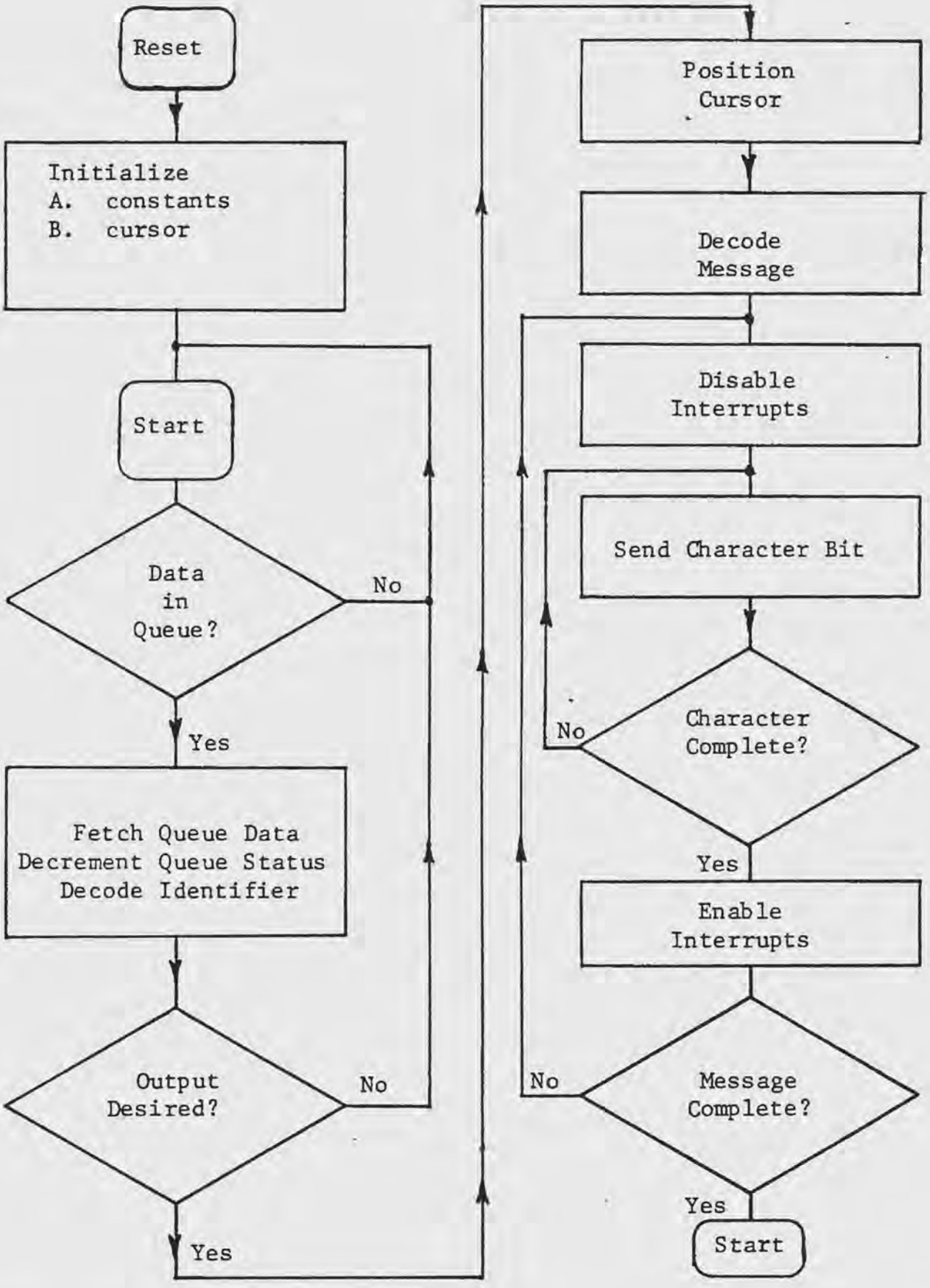


Fig. 13a. UPI-41 control program flowchart - processing loop

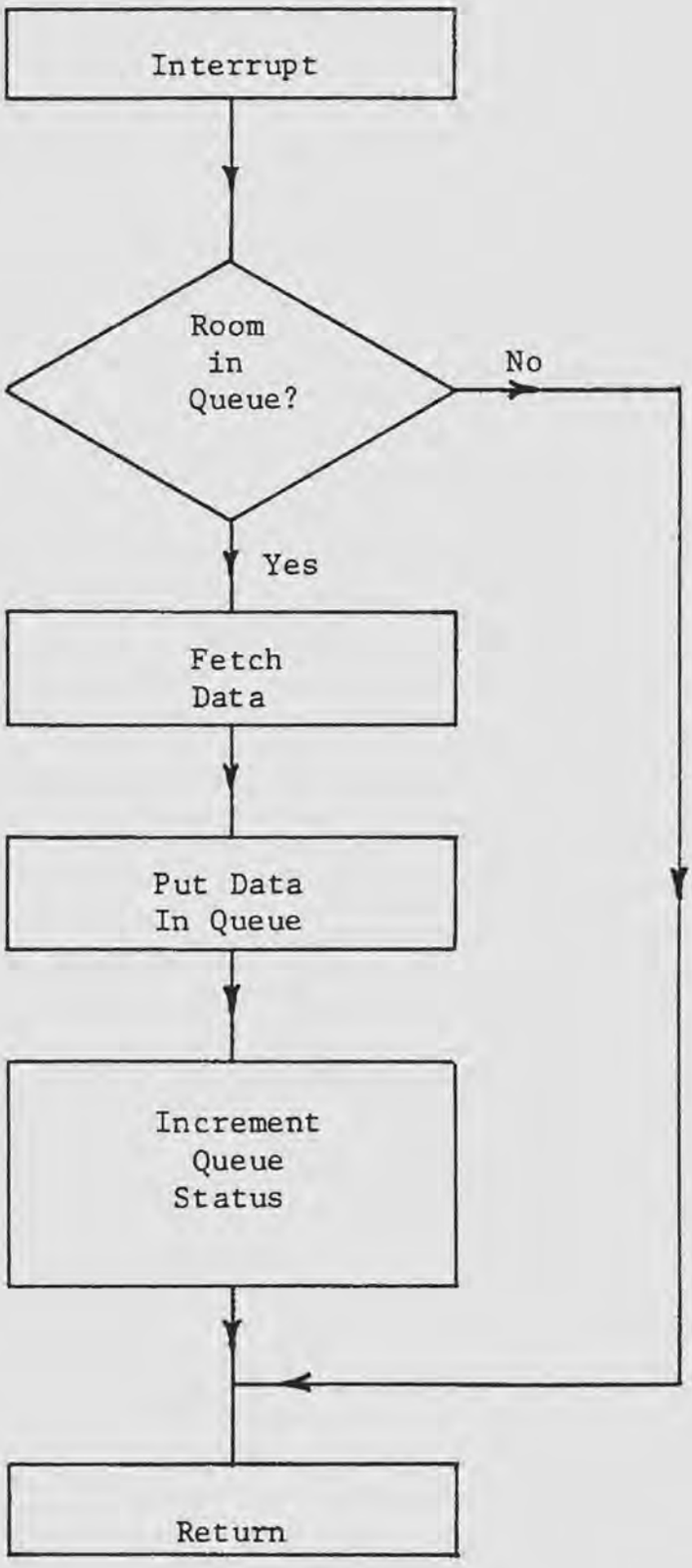


Fig. 13b. UPI-41 control program flowchart - interrupt service routine

tion procedure consists of clearing the screen and positioning the cursor in the bottom left hand corner. The screen is cleared by transmitting a special character, 01AH, to the ADM-3A (3), while the cursor is positioned using the 4 character cursor positioning sequence described previously in the controller to ADM-3A interface section.

As the final step in the initialization procedures, the UPI-41 enables itself to data reception by outputting a logic zero to port 2 line 7. This port line is connected to the UPI-41 chip select, $\overline{CS}$, input as shown in the schematic in appendix 1. Since all port lines are in the logic high state following a system reset, UPI-41 input is disabled until the output instruction is executed.

Data Reception and Storage

When data is written into the UPI-41 interface register by the SBC 80/20, an interrupt request is generated. Upon recognition of the interrupt, the interrupt vector jump at locations 3 and 4 in program memory is executed, and the interrupt service routine, lines 118 through 134 in appendix 2, is entered. The interrupt routine inputs the data from the interface register and places the data in a queue. A flowchart of the interrupt service routine appears in figure 14.

It was pointed out in the section describing the SBC 80/20 to controller interface that the UPI-41 reception rate requirement would necessitate the data queue. The necessity for the queue can be shown as follows:

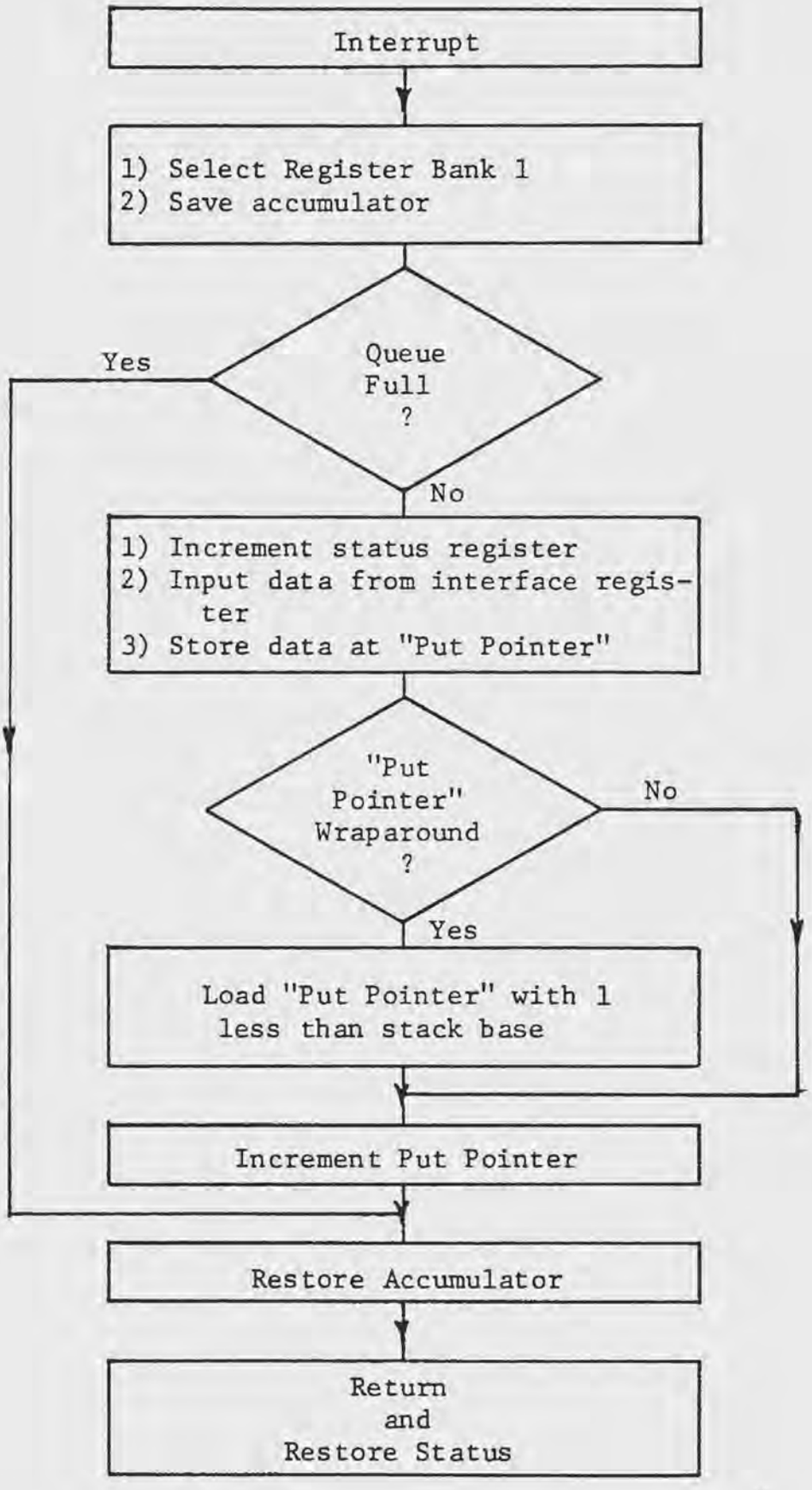


Fig. 14. Interrupt routine flowchart

Unless the 19,200 baud rate can meet the UPI-41 reception rate requirement as well as the processing rate requirement, it is necessary to provide a data queue to prevent data from being overwritten in the interface register. For this condition to be met, the 19,200 baud rate must be proportionately greater than the 11,800 minimum baud rate by at least the proportion of the reception rate requirement to the processing rate requirement, or

$$\frac{19,200}{11,800} \overset{M}{\geq} \frac{200}{60} \qquad (2)$$

as this is not true, a queue must be maintained.

To meet the storage requirements a First In First Out, or FIFO, stack is implemented in the variable data storage area of the RAM memory. See figure 12. A FIFO stack allows data to be retreived so that order of entry is preserved. The operation of a FIFO stack can be conceptualized by considering a storage mechanism where data inputs are stacked one on top of the other as they arrive, and where data removal is accomplished by pulling from the bottom. As an entry is removed, all remaining entries move down one location. This operation is illustrated in figure 15.

The problem with this implemention is in moving the remaining data entries down. For N remaining inputs, the operation requires 2N memory accesses and 5N program steps as shown below:

1. Increment pointer
2. Load data byte - first memory access
3. Decrement pointer
4. Store data byte - second memory access
5. Increment pointer

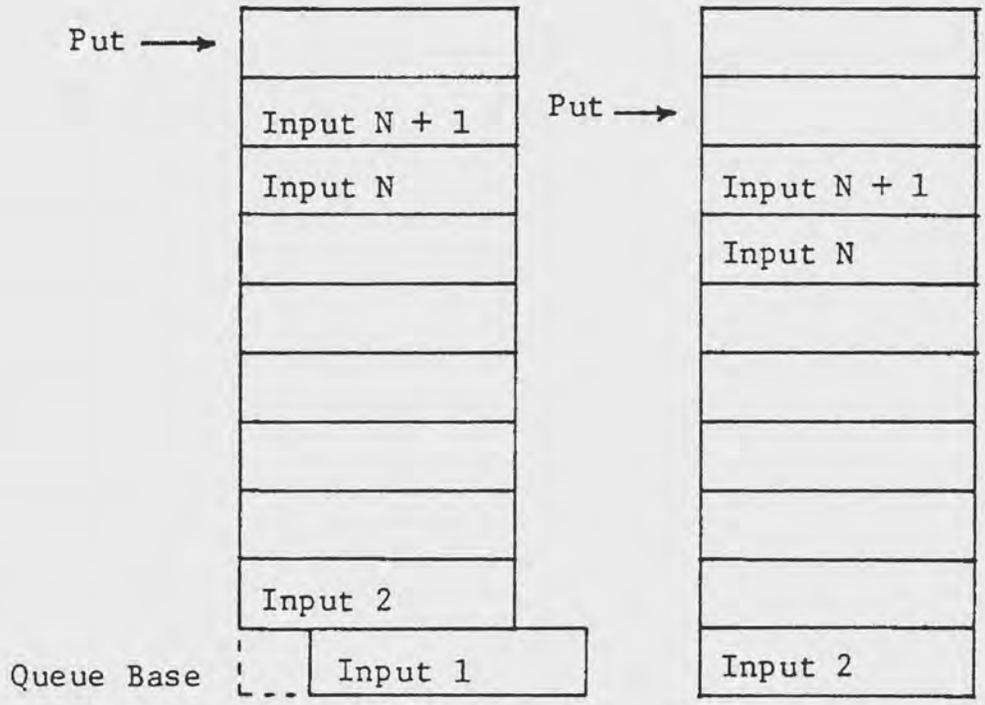


Fig. 15. Fixed base FIFO operation

A more efficient algorithm uses a "get data" pointer as well as the "put data" pointer used in the implementation above. The get data pointer allows the "bottom" of the stack to move upward as data is removed from the stack. This eliminates the necessity of moving each of the remaining inputs down. Instead, the get data pointer is incremented once each time data is removed. The put data pointer always identifies the next location available for data storage and the get data pointer identifies the location of the next value to be removed. The only problem with this implementation is that unless data memory is infinitely long, storage locations will run out at some point. This condition being unacceptable, a "top-of-stack" must be defined, and as the pointers reach the top they must wraparound. In this application the top-of-stack has been made coincident with the top of RAM, making the last location ad-

dress 63 and giving a stack size of (63 - 32) +1, or 32 locations. As each pointer reaches location 63 it is returned to location 32 instead of being incremented further. Implemented in this manner, the number of steps required for a data removal is independent of N and, for the UPI-41, has a maximum value of 5 as indicated by lines 85 through 89 of the program listing.

For either implementation some way of determining when the stack is full must be available. For the two pointer implementation, the queue full condition is easily detected by maintaining a queue status value which indicates how many entries are presently on the stack. If a check of the queue status register indicates that the queue is full additional data must be rejected to avoid overwriting of the earliest entry with the newest entry. Since the UPI-41 has been designed to meet the processing rate requirement, it follows that the maximum stack usage must be less than or equal to the number of SBC 80/20 input sources, or 5; therefore, the queue full condition can never occur in this application. Use of the queue status value in this application, then, is limited to determining when data is available on the stack. Figure 16 illustrates the operation of the moving base FIFO stack.

The put and get data pointers, the queue status, and the constants used to determine the pointer wraparound and queue full conditions are located in register bank 1. Also, since the data reception routine is entered in response to an interrupt, another bank 1 register is allocated for accumulator storage. Finally, one register is used for temporary data byte storage during computations.

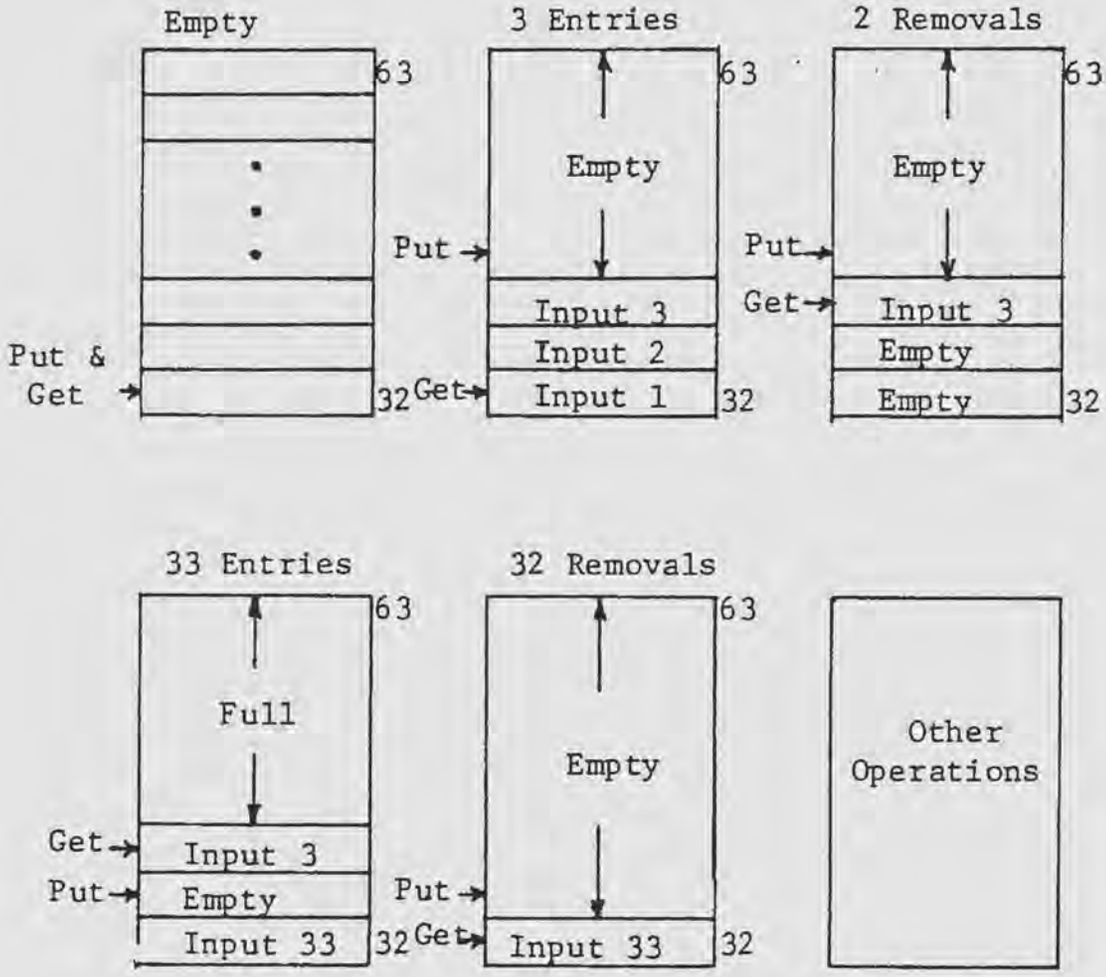


Fig. 16. Moving base FIFO operation

Registers 0 and 1 are the only locations which can serve as pointers into the variable data storage area; therefore, the get and put data pointers are defined as the contents of registers 0 and 1 respectively, the other locations are assigned arbitrarily as per figure 17.

Register 7'	Temporary Storage
Register 6'	Queue Status Con. = 224
Register 5'	Wraparound Constant = 193
Register 4'	Unused
Register 3'	Accumulator Storage
Register 2'	Queue Status
Register 1'	Put Data Pointer
Register 0'	Get Data Pointer

Fig. 17. Register bank 1 map

Data Decode and Message Transmission

Once data is placed in the queue by the interrupt service routine, a check of the queue status register, lines 80 and 81 of the program listing, will indicate that data is available for processing. The program will then enter the main program loop, line 82, where the data decode and message transmission function begins.

This section of the program can be divided into 3 segments:

1. Data access
2. Source processing
3. Message processing

Data Access

The function of the data access segment is to remove a data byte from the queue and perform the transition between register bank 1 operation and register bank 0 operation. The data removal

steps are reminiscent of the steps performed in the interrupt routine, while the bank transition is accomplished by placing the data in the accumulator and then selecting the new register bank. A flowchart is shown in figure 18.

Register bank 0 is used for the remainder of the program. All locations within this bank are assigned arbitrarily as shown in figure 19.

Register 7	Data Byte
Register 6	Message Length Constant
Register 5	Binary Source Identifier
Register 4	Linear Select Source Identifier
Register 3	Message Identifier
Register 2	Relay Counter
Register 1	Unused
Register 0	Serial Transmission Counter

Fig. 19. Register bank 0 map

Source Processing

The function of the source processing segment, lines 93 through 98, is to use the source identifier portion of the data byte to (1) determine whether a message transmission is desired and (2) position the cursor at the proper place on the ADM-3A screen. The source processing segment calls three subroutines; MASK, LOCSET, and TAB.

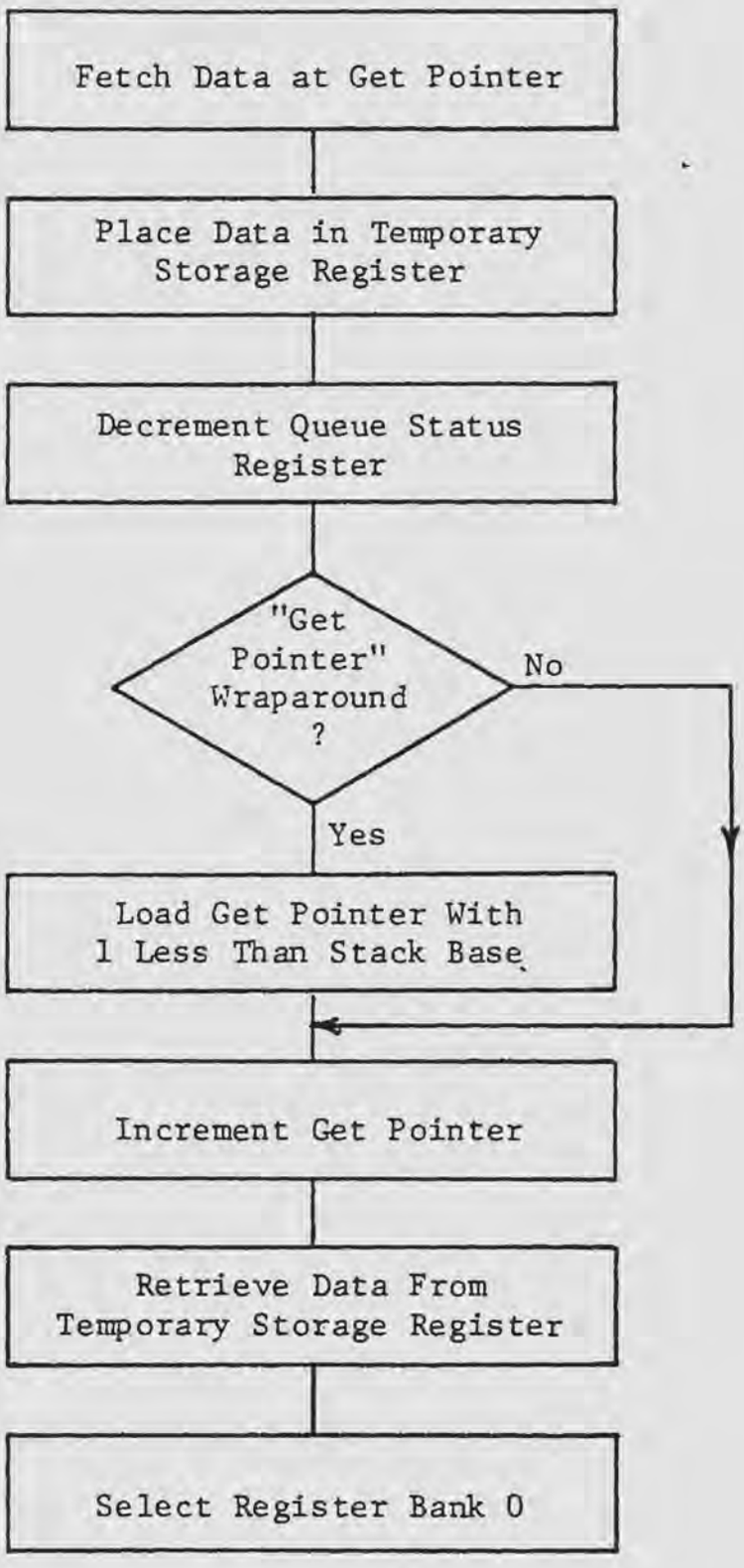


Fig. 18. Data access segment flowchart

Subroutine MASK, lines 144 through 156, coverts the binary source identifier into the linear select identifier through the use of the lookup table located at MSKDAT, line 143. The subroutine then performs the comparison with the port 2 control switch lines and sets a flag according to the result.

Subroutine LOCSET, lines 161 through 168, sends the characters which activate the cursor control logic and the Y coordinate value to the ADM-3A.

Subroutine TAB, lines 157 through 160, converts the binary source identifier into the proper X coordinate value and completes the cursor positioning sequence by transmitting the coordinate value to the ADM-3A.

Message Processing

The function of the message processing segment, lines 99 through 112, is to convert the message identifier portion of the data byte into the page 3 address of the message string, output the message string, scroll the ADM-3A display one line, and return to the queue status checking loop.

The page 3 address of the message string is produced by multiplying the binary message identifier by 16. Thus, the message identifier is converted into the starting address of a 16 character string which makes up the message. The multiplication is accomplished by swapping the high and low order nybbles of the data byte and then masking out the low order nybble. This operation is equivalent to four left shifts and, therefore, multiplies the scource

identifier by 2^4, or 16.

Subroutine STROUT, lines 169 through 175, uses the message address produced by the preceeding multiplication and the string length constant contained in register 6 to control the transmission of the 16 character message string to the ADM-3A.

The CRLF procedure, lines 107 through 110 cause the scroll of the ADM-3A display by sending the carriage return line feed combination.

Finally, register bank 1 is selected so that when the jump at line 112 occurs the register bank containing the queue status value, R2', will be addressed by the WAIT loop.

This completes the description of the control program except for the subroutine which controls character transmission. This function is accomplished by the OUTPUT subroutine, lines 176 through 191.

It was noted in the description of the clock connection, section II, that the 4.608 megahertz clock input to the UPI-41 would be used to generate the proper communication frequency. The following discussion explains this process and the operation of the OUTPUT subroutine.

Each instruction in the UPI-41 instruction set consists of either 1 or 2 instruction cycles. Each instruction cycle consists of 5 machine states and each state consists of 3 clock periods. See figure 20.

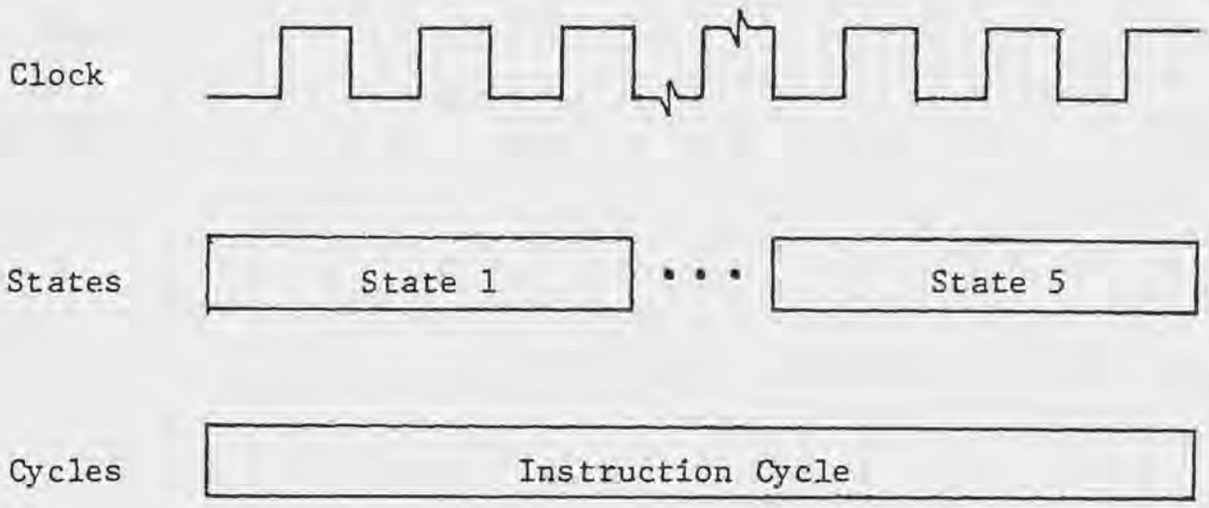


Fig. 20. UPI-41 instruction cycle

The instruction cycle execution rate, then, is 1/15 of the input clock rate, or 307,200 instruction cycles per second. The instruction cycle execution rate divided by 16 produces the serial transmission rate of 19,200 baud. Therefore, a bit interval, i.e. the time a serial bit should be present on port 1 during transmission, is exactly 16 instruction cycles. A 9 bit character can be transmitted by constructing a loop which places a new serial bit on the port 1 transmission line every 16 instruction cycles.

The OUTPUT subroutine, figure 21, expects the 7 least signifigant accumulator bits to hold the 7 bit ASCII representation of the character to be sent. As 9 bits are required to send a complete character, including the start and stop bits, the 8 bit accumulator and the carry bit are catenated to form a 9 bit register. The accumulators most significant bit and the carry bit serve as the stop and start bits respectively. Once the 9 bit register is set up with the character, the bits are sent by successively rotating the bits into the least significant bit position of the accumulator

and then outputting the accumulator to port 1.

Instructions 1, 2, and 3 set up the character, the transmission loop begins at line 180. Note that the number of instruction cycles required for each instruction in the transmission loop is shown to the right of the instruction.

For the first eight bits transmitted, program execution proceeds through the steps indicated 1 through 8. As can be verified by the reader, 16 instruction cycles are executed between bit changes.

Program flow for the final bit proceeds through the steps indicated A through E. While this sequence requires only 9 instruction cycles, analysis of the complete program shows that for any set of conditions a minimum of 8 additional instruction cycles will be required to reach the initial output instruction for a new character. Thus, a minimum of 9 + 8, or 17, cycles will be executed exceeding the minimum of 16 by 1 cycle. But, as there is no maximum length for the stop bit since its level corresponds to the non active, or "marking" state, the value 17 is acceptable.

The instruction executed just prior to entry into the bit transmission loop disables interrupts, while the instruction just before the return reenables them. Interrupts must be disabled during transmission of a character since the occurance of an interrupt service routine would insert extra instruction cycles, thereby destroying the integrity of the software timing loop.

```
012E 97          176 OUTPUT: CLR   C          ;CLEAR CARRY BIT
012F E7          177         RL    A          ;POSITION START BIT
0130 A7          178         CPL   C          ;SET CARRY BIT
0131 15          179         DIS   I          ;DISABLE INTERRUPTS
0132 39      A 1 180 LOOP1:  OUTL  P1,A       2;PUT SERIAL BIT ON TRANSMIT LINE
0133 E839    B 2 181         DJNZ  R0,CONTIN  2;JUMP TO CONTINUE IF NOT DONE
0135 B809    C   182         MOV   R0,#09D    2;RELOAD BIT COUNTER
0137 05      D   183         EN    I          1;ENABLE INTERRUPTS
0138 83      E   184         RET              2;RETURN FROM SUBROUTINE
0139 67          3 185 CONTIN: RRC   A          1;ROTATE NEXT BIT INTO TRANSMIT POSITION
013A 00          4 186         NOP              1;EVEN CYCLE COUNT
013B BA01        5 187         MOV   R2,#01     2;SET DELAY LENGTH
013D 3441        6 188         CALL  DELAY      2;CALL DELAY SUBROUTINE
013F 2432        9 189         JMP   LOOP1      2;CONTINE LOOP EXECUTION
0141 EA41        7 190 DELAY:  DJNZ  R2,DELAY   2;VARIABLE DELAY DEPENDING ON R2
0143 83          8 191         RET              2;RETURN FROM SUBROUTINE
```

Fig. 21. UPI-41 character transmission subroutine

As a concluding remark on the UPI-41 control program, it is noted that starting on page 51 of the listing a sample set of message strings is shown.

The program listing refered to throughout this section is the assembly listing produced during the assembly of the UPI-41 control program source file. This version of the program was used for the system evaluation to be presented in the next section.

V. SYSTEM EVALUATION

In section II equation (1), the minimum UPI-41 to ADM-3A serial transmission rate was computed by considering only the character string transmission time. It was assumed at that point that processing done in decoding the input data would be insignificant compared to the processing done within the serial transmission routine. The validity of this assumption can be evaluated by determining from the control program the percentage of instruction cycles which are executed outside of the serial transmission routine.

As determined in section IV, sixteen instruction cycles are required per serial bit transmitted. Since 22 characters at 9 bits per character are sent for each input, the number of transmission instruction cycles executed for each string transmission is

16 instruction cycles/bit x 9 bits/character x 22 characters/string
or 3168 instruction cycles/string (3)

Analysis of the UPI-41 program shows that 236 instruction cycles are executed external to the serial transmission routine. Thus, the percentage of instruction cycles executed outside of the serial transmission routine is

$$\frac{\text{236 instruction cycles}}{\text{3168 instruction cycles}} \times 100 \text{ percent, or } 7.45 \text{ percent} \qquad (4)$$

The assumption, then, gave a minimum baud rate within 10% of the exact figure.

To meet the UPI-41 processing rate requirement requires that the minimum baud rate of 11,880 computed in equation (1) be adjusted upward by 7.45 percent. The new minimum baud rate becomes

$$11{,}880 \text{ bits/sec} + (.0745 \times 11{,}880 \text{ bits/sec}), \text{ or } 12{,}765 \text{ bits/sec} \quad (5)$$

However, since the baud rate chosen was already 19,200 bits per second due to the ADM-3A receiving rate constraints, no program or hardware adjustments are necessary.

Transmission at 19,200 baud instead of the minimum baud provides an immediate system expansion capability of

$$\frac{19{,}200 \text{ bits/sec} - 12{,}765 \text{ bits/sec}}{12{,}765 \text{ bits/sec}} \times 100 \text{ percent} \quad (6)$$

or approximately 50 percent. This expansion can be in terms of the number of SBC 80/20 input sources or the source input rate, see figure 5. If the source input rate is maintained at 12 inputs per second, the number of sources could be expanded to 7.5, or realistically 7. Conversely, if the number of sources is taken as a constant, the input rate can be expanded from 12 inputs per second to 18 inputs per second.

The expansion to 18 inputs per second would require no additional system expansion, while the expansion to 7 inputs would require either a wider display screen or a slight overlapping of the message columns. Overlapping the columns would be preferable since

a wider screen would necessitate a new display device.

It was shown in section II that if the UPI-41 reception rate requirement was to be met, a data queue would have to be maintained. While a necessary requirement, however, the queue is not sufficient to guarantee that the reception rate requirement will be met. The guarantee must be provided by insuring that the worst case interrupt response time meets the requirement. The worst case response occurs when an input occurs just as interrupts are disabled following entry into the serial transmission loop. The interrupt response time for this condition will be essentially equal to the character transmission time. The worst case reception rate will be one over this time and can be represented as

$$\frac{1}{\frac{1}{19{,}200 \text{ bits/sec}} \times 9 \text{ bits/transfer}} \qquad (7)$$

or 2133 transfers per second. This value exceeds the required UPI-41 data reception rate of 200 transfers per second by better than an order of magnitude.

Comparing the above order of magnitude difference with the 50% difference computed in relation to the UPI-41 processing rate requirement, it is seen that the constraints on system expansion are primarily associated with data processing rather than data reception, and, therefore, with the serial transmission rate. System expansion beyond 7 sources or 18 inputs per second would only be possible by increasing the serial transmission rate. As 19,200 baud is the highest transmission rate supported by the ADM-3A, any expan-

sion would require going to a different display device.

In terms of the UPI-41, the limits on system expansion are related to its maximum serial transmission rate, the input identifier field of the data byte, and the queue size.

If the input clock rate to the UPI-41 was increased to 5.76 megahertz, the highest clock rate which is both within the 1 - 6 megahertz operating range and an integral multiple of the standard communication frequencies, the baud rate could be quadrupled to 76,800 baud. This baud rate corresponds to a serial bit time equivalent to 5 instruction cycles at the 2.604 microseconds per instruction cycle established by the 5.76 megahertz clock. By decreasing to a minimum the number of instructions executed in the bit generation loop of the OUTPUT subroutine as follows:

```
Loop 1:   OUTL   P1, A        ;2 cycles
          RRC    A            ;1 cycle
          DJNZ   R0, Loop1    ;2 cycles
```

the 5 instruction cycle requirement can be met. For the 76,800 bit per second transmission rate, the system expansion percentage becomes:

$$\frac{76{,}800 \text{ bits/sec} - 12{,}765 \text{ bits/sec}}{12{,}765 \text{ bits/sec}} \times 100 \text{ percent} \qquad (8)$$

or approximately 500 percent.

Now, the number of sources could be increased to 5 + (5 x 5) or 30, or the maximum input rate could be increased to 12 + (12 x 5), or 72 inputs per second. The changes required for the

increased input rate would be the alteration of the output subroutine to the form shown above.

An increase in the number of sources to 30, however, must be checked for plausability against two other system constraints: the queue size, and the source identification field of the SBC 80/20 to controller data byte.

The queue size of the UPI-41 is 32 locations; thus, even if it is assumed that the queue could be filled before any input could be completely processed, the minimum number of inputs that could be handled would be the queue size plus the entry being processed, or 33. Since the baud rate already limits the input sources to 30, this is not a limiting factor. The source identification field is currently 3 bits wide as necessitated by 5 sources allowing a possible identification of 2^3 or 8 sources, see figure 4. If the undedicated bit is added to the source identifier field, then 2^4, or 16 sources could be supported. As this value is less than the limitation imposed by the serial transmission rate constraint, the source identification field becomes the limiting factor on input source expansion.

While not directly related to the scope of this thesis, it is noted that expansion of the source input rate is a real possibility as other input sources could be easily attached which have higher characteristic input rates. For 5 source inputs, the SBC 80/20 is limited to servicing a source input rate of 40 inputs per second. Thus, with a display device having a higher serial reception rate, the controller could easily handle the maximum source

input rate imposed by the SBC 80/20. Expansion to greater than 5 sources would require rather extensive hardware and software changes within the SBC 80/20. For the source input rate of 12 inputs per second, the SBC 80/20 could handle 16 input sources. This value also compares favorably with the maximum capability imposed by the UPI-41, also equal to 16 sources.

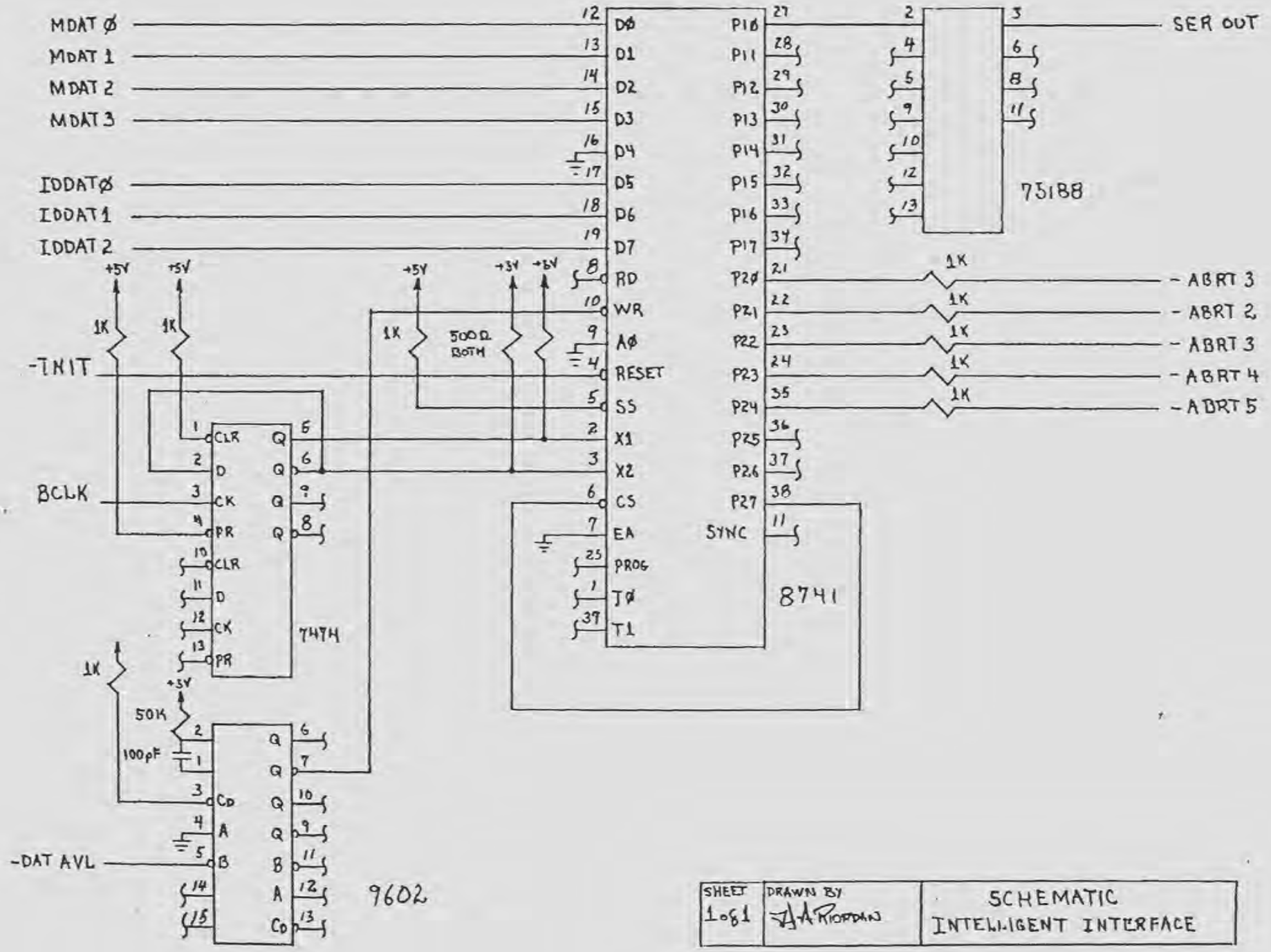
VI. CONCLUSION

The use of the UPI-41 as described in this thesis follows a current trend in the design of computer control systems. This trend is characterized by the spreading of the "intelligence" throughout the entire system. Systems designed in this fashion are grouped under the general classification of "distributed processor systems". The present application can be further classified as a master/slave distributed processor system, where the SBC 80/20 computer functions as the master and the UPI-41 functions as the slave. Distributed processor systems provide for a certain degree of parallelism within a control task, and thereby increase the operating speed of the overall system. As the cost of processing elements continues to decrease, distributed processor systems will become increasingly prevalent in computer control applications.

The design and construction of the intelligent controller system was completed during the summer/fall quarter of 1978. In many tests since that time the system has performed as expected.

APPENDIX 1

CONTROLLER SCHEMATIC



SHEET	DRAWN BY	SCHEMATIC
1081	J.A. RIORDAN	INTELLIGENT INTERFACE

APPENDIX 2

UPI-41 CONTROL PROGRAM

```
ASM48 :F1:UPI41B.SRC MACROFILE DEBUG MOD41 TITLE('27 FEB 79')


ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0
27 FEB 79

 LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 ;      ************************************************************
                    2 ;      *                                                          *
                    3 ;      *                                                          *
                    4 ;      *              UPI-41 CONTROL PROGRAM                      *
                    5 ;      *                                                          *
                    6 ;      *                                                          *
                    7 ;      ************************************************************
                    8 ;
                    9 ;
                   10 ;
                   11 ;
                   12 ;      REGISTER BANK 0
                   13 ;
                   14 ;
                   15 ;      REGISTER 0(R0)  9 BIT SERIAL TRANSMISSION COUNTER
                   16 ;      REGISTER 1(R1)  UNUSED
                   17 ;      REGISTER 2(R2)  COUNT FOR VARIABLE DELAY
                   18 ;      REGISTER 3(R3)  CHARACTER STRING INDEX REGISTER
                   19 ;      REGISTER 4(R4)  SOURCE IDENTIFIER (LINEAR SELECT)
                   20 ;      REGISTER 5(R5)  SOURCE IDENTIFIER (BINARY)
                   21 ;      REGISTER 6(R6)  MESSAGE STRING LENGTH CONSTANT
                   22 ;      REGISTER 7(R7)  PARALLEL DATA TRANSFER
                   23 ;
                   24 ;      REGISTER BANK 1
                   25 ;
                   26 ;
                   27 ;      REGISTER 0(R0)  GET DATA POINTER
                   28 ;      REGISTER 1(R1)  PUT DATA POINTER
                   29 ;      REGISTER 2(R2)  QUEUE STATUS
                   30 ;      REGISTER 3(R3)  ACCUMULATOR STORAGE
                   31 ;      REGISTER 4(R4)  UNUSED
                   32 ;      REGISTER 5(R5)  WRAPAROUND CONSTANT=193D
                   33 ;      REGISTER 6(R6)  QUEUE STATUS CONSTANT=224D
                   34 ;      REGISTER 7(R7)  TEMPORARY DATA WORD STORAGE
                   35 ;
                   36 ;
                   37 ;      PORT 1   SERIAL TRANSMISSION ON BIT 0
```

```
                    38 ;        PORT 2   LINES 0-4 USED AS A MASK INPUT TO INHIBIT TEXT
                    39 ;                 STRING OUTPUT.  LINE 7 USED TO ENABLE CHIP SELECT.
                    40 ;
                    41 ;
                    42 ;
                    43 ;
0000                44          ORG 0
0000 040A           45          JMP     10D             ;PRESERVE INTERRUPT VECTORS
0003                46          ORG 3D
0003 0458           47 EXTINT:  JMP     INROUT          ;JUMP TO INTERRUPT ROUTINE
                    48 ;
                    49 ;
000A                50          ORG 10D
000A B809           51 INIT:    MOV     R0,#09D         ;INIT BIT TRANSMISSION COUNTER
000C 231A           52          MOV     A,#1AH          ;ASCII CHAR TO CLEAR CRT SCREEN
000E 342E           53          CALL    OUTPUT
0010 23FF           54          MOV     A,#0FFH         ;START OF NESTED DELAY
0012 BAFF           55 LOOP:    MOV     R2,#0FFH
0014 3441           56          CALL    DELAY
0016 07             57          DEC     A
0017 9612           58          JNZ     LOOP            ;END OF NESTED DELAY
0019 3419           59          CALL    LOCSET          ;SET UP CRT TO ACCEPT X COORD VALUE
001B 2320           60          MOV     A,#20H          ;X VALUE FOR COLUMN 1
001D 342E           61          CALL    OUTPUT          ;ROUTINE TO SEND ASCII CHARACTER
001F D5             62          SEL     RB1
0020 B820           63          MOV     R0,#32D         ;INITIAL VALUE FOR READ MEMORY POINTER
0022 B920           64          MOV     R1,#32D         ;INITIAL VALUE FOR WRITE MEMORY POINTER
0024 BA00           65          MOV     R2,#0           ;CLEAR QUEUE STATUS REGISTER
0026 BEE0           66          MOV     R6,#224D        ;224 + 32 AVAILABLE LOCATIONS IN RAM
                    67                                  ;= 256 => OVERFLOW
0028 BDC1           68          MOV     R5,#193D        ;193 + 63(LAST RAM ADDRESS) = 256 =>
                    69                                  ;OVERFLOW
002A 237F           70          MOV     A,#7FH          ;NUMBER WILL KEEP LINES 0-6 AS INPUTS
                    71                                  ;AND WILL ENABLE THE CHIP SELECT WHICH
                    72                                  ;IS TIED TO LINE 7
002C 3A             73          OUTL    P2,A
002D 05             74          EN      I               ;ENABLE EXTERNAL INTERRUPTS
                    75 ;
                    76 ;
                    77 ;
                    78 ;
                    79 ;
002E FA             80 WAIT:    MOV     A,R2            ;GET QUEUE STATUS
002F C62E           81          JZ      WAIT            ;IF QUEUE EMPTY NO ACTION
0031 F0             82 START:   MOV     A,@R0           ;GET DATA FROM RAM LOCATION
```

```
0032 AF          83          MOV     R7,A        ;STORE DATA
0033 CA          84          DEC     R2          ;DECREMENT QUEUE STATUS REGISTER
0034 FD          85          MOV     A,R5        ;193 DECIMAL
0035 68          86          ADD     A,R0        ;CHECK FOR LAST ACCESS BEING @ TOP OF RAM
0036 963A        87          JNZ     CONT
0038 B81F        88          MOV     R0,#31D     ;ONE LESS THAN BOTTOM OF RAM
003A 18          89 CONT:    INC     R0          ;INCREMENT GET DATA POINTER
003B FF          90          MOV     A,R7        ;RETRIEVE DATA
003C C5          91          SEL     RB0
003D AF          92          MOV     R7,A        ;STORE DATA
003E 3485        93          CALL    MASK        ;CHECK TO SEE IF OUTPUT DESIRED
0040 B658        94          JF0     ESCAPE      ;IF FLAG SET WAIT FOR NEW DATA
0042 3419        95          CALL    LOCSET      ;SET UP ADM TO ACCEPT X-COORD VALUE
0044 FD          96          MOV     A,R5        ;GET DATA ID FROM PROCEDURE MASK STORAGE
                 97                              ;LOCATION
0045 3414        98          CALL    TAB         ;TAB OVER TO LOCATION CORRES TO DATA #
0047 FF          99          MOV     A,R7        ;RETRIEVE 80/20 DATA
0048 47         100          SWAP    A           ;PUT CODE FOR MESSAGE ACCESS
                101                              ;IN UPPER 4 BITS TO ALLOW ACCESS TO
                102                              ;16 MEMORY LOCATIONS PER SHOT TYPE
0049 53F0       103          ANL     A,#0F0H     ;MASK OUT LOW ORDER BITS
004B AB         104          MOV     R3,A        ;STORE RELATIVE ADDRESS OF CHAR STRING
004C BE10       105          MOV     R6,#16D     ;STRING LENGTH COUNTER
004E 3426       106          CALL    STROUT      ;PROCEDURE TO OUTPUT ASCII STR
0050 230A       107 CRLF:    MOV     A,#0AH      ;LINE FEED
0052 342E       108          CALL    OUTPUT
0054 230D       109          MOV     A,#0DH      ;CARRIAGE RETURN
0056 342E       110          CALL    OUTPUT
0058 D5         111 ESCAPE:  SEL     RB1         ;RETURN TO CORRECT REG BANK FOR WAIT LOOP
0059 042E       112          JMP     WAIT
                113 ;
                114 ;
                115 ;
                116 ;
                117 ;
005B D5         118 INROUT:  SEL     RB1         ;INTERRUPT REG BANK
005C AB         119          MOV     R3,A        ;SAVE ACCUMULATOR
005D FE         120          MOV     A,R6        ;224D
005E 6A         121          ADD     A,R2
005F C668       122          JZ      QUEFUL      ;CHECK FOR QUEUE FULL
0061 1A         123          INC     R2          ;INCREMENT QUEUE STATUS REGISTER
0062 22         124          IN      A,DBB       ;INPUT DATA
                125                              ;FROM SR INTERRUPT STORE FF
0063 A1         126          MOV     @R1,A       ;STORE NEW DATA
0064 FD         127          MOV     A,R5        ;193D
```

```
0065 69          128          ADD     A,R1          ;CHECK TO SEE IF STORE WAS
                 129                                ;IN LAST AVAILABLE RAM LOCATION
0066 966R        130          JNZ     CONT1         ;IF NOT THEN CONTINUE
0068 B91F        131          MOV     R1,#31D       ;BOTTOM OF QUEUE
006A 19          132 CONT1:   INC     R1            ;INCREMENT PUT DATA REGISTER
006B FB          133 QUEFUL:  MOV     A,R3          ;RESTORE ACCUMULATOR
006C 93          134          RETR                  ;RETURN FROM INTERRUPT
                 135                                ;& REENABLE INTERRUPTS
                 136 ;
                 137 ;
                 138 ;  SUBROUTINES IN SECOND PAGE OF MEMORY
                 139 ;
                 140 ;
                 141 ;
0100             142          ORG 256D
0100 01          143 MSKDAT:  DB      1D,2D,4D,8D,16D
0101 02
0102 04
0103 08
0104 10
0105 53E0        144 MASK:    ANL     A,#0E0H       ;MASK OUT 5 LOW ORDER BITS
0107 47          145          SWAP    A             ;DATA ID IN BITS 1,2,3
0108 77          146          RR      A             ;IN BITS 0,1,2
0109 AD          147          MOV     R5,A          ;STORE DATA CODE
010A 07          148          DEC     A             ;DATA CODE(FILE) 0-4
010B A3          149          MOVP    A,@A          ;GET LOOKUP VALUE FOR CURRENT DATA
010C AC          150          MOV     R4,A          ;STORE VALUE
010D 0A          151          IN      A,P2          ;GET DATA MASK
010E 5C          152          ANL     A,R4
010F 85          153          CLR     F0
0110 9613        154          JNZ     CONT2         ;JUMP IF DATA NOT MASKED
0112 95          155          CPL     F0            ;SET FLAG INDICATING MASK
0113 83          156 CONT2:   RET                   ;RETURN FROM SUBROUTINE
0114 17          157 TAB:     INC     A             ;CREATE CORRECT DIGIT FOR HIGH BYTE
0115 47          158          SWAP    A             ;PUT IN HIGH BYTE
0116 342E        159          CALL    OUTPUT
0118 83          160          RET                   ;RETURN FROM SUBROUTINE
0119 231B        161 LOCSET:  MOV     A,#1BH        ;ASCII ESCAPE (REF ADM MANUAL
                 162                                ;RE CURSOR POSITIONING)
011B 342E        163          CALL    OUTPUT
011D 233D        164          MOV     A,#3DH        ;ASCII EQUALS
011F 342E        165          CALL    OUTPUT
0121 2337        166          MOV     A,#37H        ;ROW 24 OF ADM TERMINAL
0123 342E        167          CALL    OUTPUT
0125 83          168          RET                   ;RETURN FROM SUBROUTINE
```

```
0126 FB          169 STROUT: MOV     A,R3          ;RETRIEVE PAGE 3 ADDRESS OF
                 170                               ;ASCII STRING
0127 E3          171         MOVP3   A,@A          ;GET ASCII CHARACTER
0128 1B          172         INC     R3            ;NEXT CHARACTER
0129 342E        173         CALL    OUTPUT
012B EE26        174         DJNZ    R6,STROUT     ;HAVE ALL CHAR BEEN OUTPUT
012D 83          175         RET                   ;RETURN FROM SUBROUTINE
012E 97          176 OUTPUT: CLR     C             ;CLEAR CARRY BIT
012F E7          177         RL      A             ;POSITION START BIT
0130 A7          178         CPL     C             ;SET CARRY BIT
0131 15          179         DIS     I             ;DISABLE INTERRUPTS
0132 39          180 LOOP1:  OUTL    P1,A          ;PUT SERIAL BIT ON TRANSMIT LINE
0133 E839        181         DJNZ    R0,CONTIN     ;JUMP TO CONTINUE IF NOT DONE
0135 B809        182         MOV     R0,#09D       ;RELOAD BIT COUNTER
0137 05          183         EN      I             ;ENABLE INTERRUPTS
0138 83          184         RET                   ;RETURN FROM SUBROUTINE
0139 67          185 CONTIN: RRC     A             ;ROTATE NEXT BIT INTO TRANSMIT POSITION
013A 00          186         NOP                   ;EVEN CYCLE COUNT
013B BA01        187         MOV     R2,#01        ;SET DELAY LENGTH
013D 3441        188         CALL    DELAY         ;CALL DELAY SUBROUTINE
013F 2432        189         JMP     LOOP1         ;CONTINE LOOP EXECUTION
0141 EA41        190 DELAY:  DJNZ    R2,DELAY      ;VARIABLE DELAY DEPENDING ON R2
0143 83          191         RET                   ;RETURN FROM SUBROUTINE
                 192 ;
                 193 ;
                 194 ; TEXT STRINGS LOCATED IN PAGE 3 OF MEMORY
                 195 ;
                 196 ;
0300             197         ORG 300H
0300 4D455353    198         DB      'MESSAGE ZERO    '
0304 41474520
0308 5A45524F
030C 20202020
0310             199         ORG 310H
0310 4D455353    200         DB      'MESSAGE ONE     '
0314 41474520
0318 4F4E4520
031C 20202020
0320             201         ORG 320H
0320 4D455353    202         DB      'MESSAGE TWO     '
0324 41474520
0328 54574F20
032C 20202020
0330             203         ORG 330H
0330 4D455353    204         DB      'MESSAGE THREE   '
```

```
0334 41474520
0338 54455245
033C 45202020
0340                205         ORG 340H
0340 4D455353       206         DB      'MESSAGE FOUR    '
0344 41474520
0348 464F5552
034C 20202020
0350                207         ORG 350H
0350 4D455353       208         DB      'MESSAGE FIVE    '
0354 41474520
0358 46495645
035C 20202020
0360                209         ORG 360H
0360 4D455353       210         DB      'MESSAGE SIX     '
0364 41474520
0368 53495820
036C 20202020
0370                211         ORG 370H
0370 4D455353       212         DB      'MESSAGE SEVEN   '
0374 41474520
0378 53455645
037C 4E202020
0380                213         ORG 380H
0380 4D455353       214         DB      'MESSAGE EIGHT   '
0384 41474520
0388 45494748
038C 54202020
0390                215         ORG 390H
0390 4D455353       216         DB      'MESSAGE NINE    '
0394 41474520
0398 4E494E45
039C 20202020
03A0                217         ORG 3A0H
03A0 4D455353       218         DB      'MESSAGE TEN     '
03A4 41474520
03A8 54454E20
03AC 20202020
03B0                219         ORG 3B0H
03B0 4D455353       220         DB      'MESSAGE ELEVEN  '
03B4 41474520
03B8 454C4556
03BC 454E2020
03C0                221         ORG 3C0H
03C0 4D455353       222         DB      'MESSAGE TWELVE  '
```

```
03C4 41474520
03C8 5457454C
03CC 56452020
03D0                223         ORG 3D0H
03D0 4D455353       224         DB      'MESSAGE THIRTEEN'
03D4 41474520
03D8 54484952
03DC 5445454E
03E0                225         ORG 3E0H
03E0 4D455353       226         DB      'MESSAGE FOURTEEN'
03E4 41474520
03E8 464F5552
03EC 5445454E
03F0                227         ORG 3F0H
03F0 4D455353       228         DB      'MESSAGE FIFTEEN '
03F4 41474520
03F8 46494654
03FC 45454E20
                    229         END

USER SYMBOLS
CONT   003A    CONT1  006A    CONT2  0113    CONTIN 0139
CRLF   0050    DELAY  0141    ESCAPE 0058    EXTINT 0003
INIT   000A    INROUT 005B    LOCSET 0119    LOOP   0012
LOOP1  0132    MASK   0105    MSKDAT 0100    OUTPUT 012E
QUEFUL 006B    START  0031    STROUT 0126    TAB    0114
WAIT   002E

ASSEMBLY COMPLETE,  NO ERRORS
```

APPENDIX 3

ALTERNATE BAUD RATE GENERATION SCHEME

APPENDIX 3

This appendix describes the use of the UPI-41 8 bit timer as an alternate way of generating the bit times for serial transmission.

The UPI-41 timer, figure A3-1, can be programmed to operate in a number of different modes:

1. Internally clocked - the timer counts up one every 32 UPI-41 instruction cycles
2. Externally clocked - the timer count rate is controlled by an external clock source. In this mode the maximum count rate is once every three UPI-41 instruction cycles
3. Set flag on overflow - in this mode the timer flag, TF, will be set when the counter overflows. The timer flag can be tested by conditional branch instructions
4. Interrupt on overflow - an interrupt request is generated on timer overflow. If the timer interrupt is enabled a call to location 7 of program memory is executed

Operating the UPI-41 timer in the "interrupt on overflow mode" permits the UPI-41 to execute a processing task and transmit serial data concurrently. The rationale for operating the UPI-41 in this manner is to increase the system throughput by using the instruction cycles which are wasted in the software timing implementation to do other processing tasks. The following discussion describes some of the analysis necessary to operationalize the above concept within the UPI-41 control program. Two key points will be made: (1) the gain in system throughput for this applica-

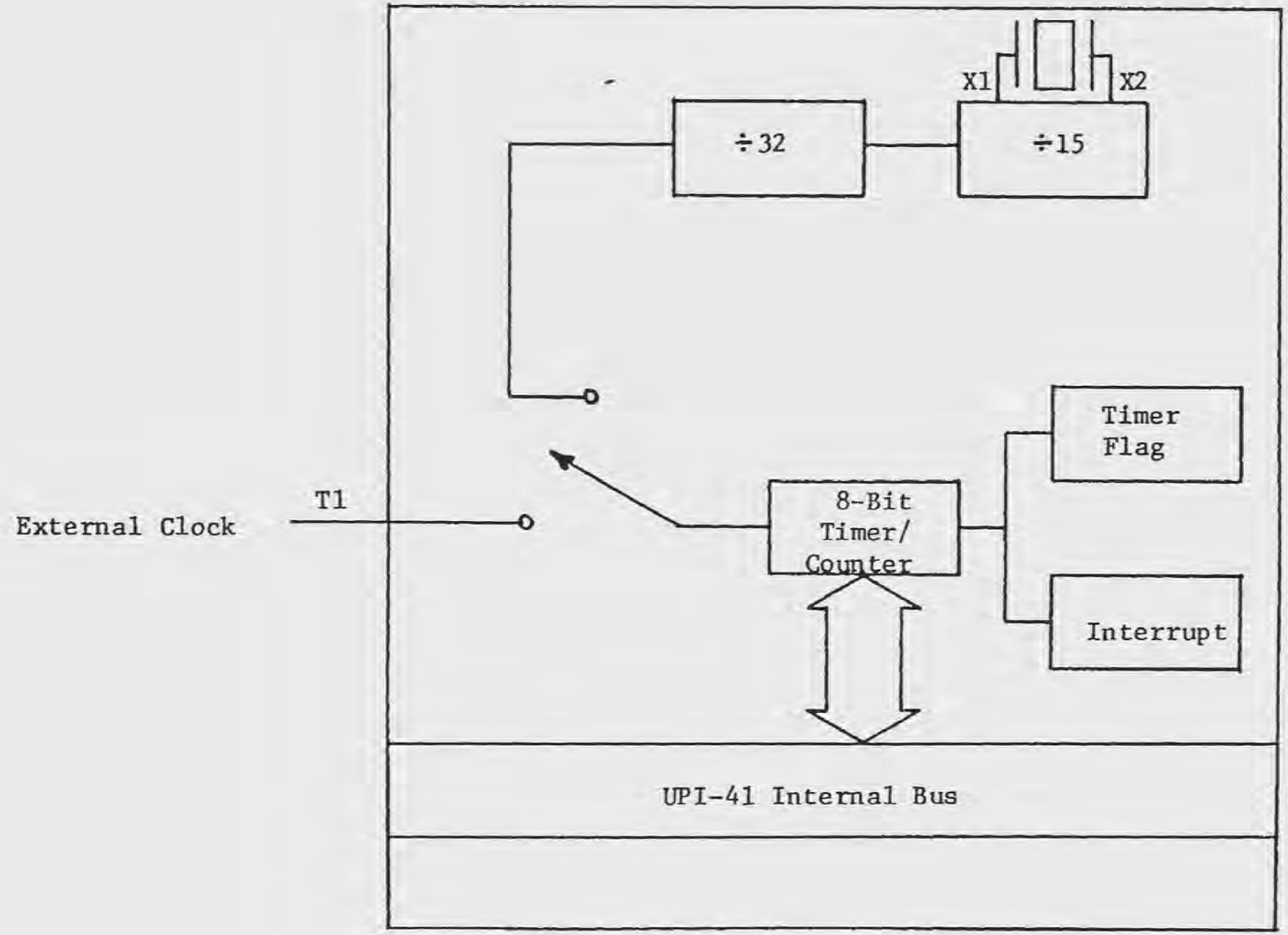


Fig. A3-1. UPI-41 Timer/counter

tion would be very small, and (2) there will be an increase in the probability of data transmission errors from close to zero for the software timing implementation to some unknown but fairly significant probability for the timer/interrupt implementation.

Consider the case where the processor and the timer are operating synchronously at their maximum respective rates. This condition corresponds to a UPI-41 input clock frequency of 6 megahertz and a timer input frequency of 133.3 kilohertz, resulting in a UPI-41 instruction cycle period of 2.5 microseconds per instruction cycle and a timer count rate of one count per 3 instruction cycles, or one count per 7.5 microseconds (4). Synchronization of the instruction cycle clock with the timer clock forces all timer interrupts to occur at the same point within an instruction cycle, while operating at the maximum permissable clock rates gives the finest time resolution. Both of these conditions appear desirable for transmission using the timer/interrupt technique.

The timer/interrupt transmission technique works by placing the transmission control loop within the timer interrupt service routine. When a character is available for transmission, it would be placed in a character storage register within the register bank used for the interrupt routine. Character transmission whould be initialized by calling the interrupt service routine directly, and the remaining bits of the character would be sent as timer interrupts occurred. There exist two cases where the timer/interrupt technique can be used. The first case requires that there be sufficient processing tasks between all character transmissions so

that the processor does not end up waiting for the previous character to be transmitted. This condition is not met in the present application, as all processing is done before character transmission begins. The second case is where the characters to be output are placed in a queue. In this way the transmission of characters can be independent of the program arrangement. For either implementation the details of the interrupt routine will be essentially identical, with the queued implementation requiring a couple of instructions to check the status of the queue. Assuming either implementation, the details of the interrupt routine can be inspected. The first issue addressed will be the timing involved, after which a discussion of the number of program steps required for implementation will be presented. For transmission at 19,200 bits per second, the bit transmission time is 1/19,200, or 52.08 microseconds, corresponding to a time interval in instruction cycles of 52.08 microseconds x 1/2.5 microseconds per instruction cycle, or 20.8 instruction cycles. The interrupt routine must be set up, then, so that bit changes will occur at intervals of 20.8 instruction cycles. As the timer can only generate interrupts in integral multiples of the instruction cycle, the best approximation to this time will be off by .2 instruction cycles. Over a 9 bit character this will result in an error of 9 x .2, or 1.8 instruction cycles, corresponding to a time shift in the final bit sent of 1.8 instruction cycles x 2.5 microseconds per instruction cycle, or 4.5 microseconds, see figure A3-2. Since the ADM-3A reads the data bits at their nominal center, however, it is unlikely that this baud rate/interrupt rate

"mismatch" alone would cause data misreads.

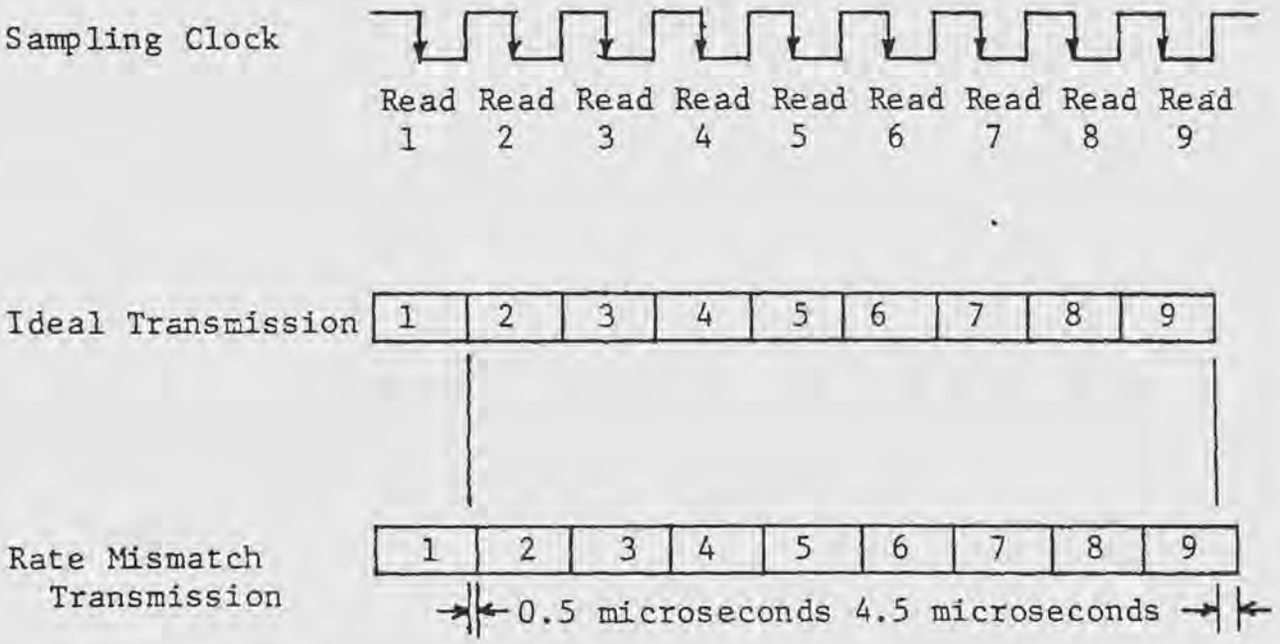


Fig. A3-2. Rate mismatch transmission

In addition to the rate mismatch error, another error arises due to the inability to predict where within an instruction execution the timer interrupt will occur. Even though an interrupt is constrained, through synchronization with the instruction cycle execution rate, to occur at the same point within an instruction cycle, it is not known whether it will occur during a one cycle or a two cycle instruction. This condition leads to an additional variation of one instruction cycle per bit interval generated. If the timer interrupt routine is set up based on the assumption that all interrupts occur during 1 cycle instructions, and the worst case condition occurs, all interrupts occur during 2 cycle instructions, then a 2.5 microseconds/bit x 9 bits/character, or 22.5 microseconds/character error will result. If the rate mismatch error is included a total error of 22.5 + 4.5, or 27 microseconds/character will occur. As shown in figure A3-3 this condition

would cause a misread of the final bit.

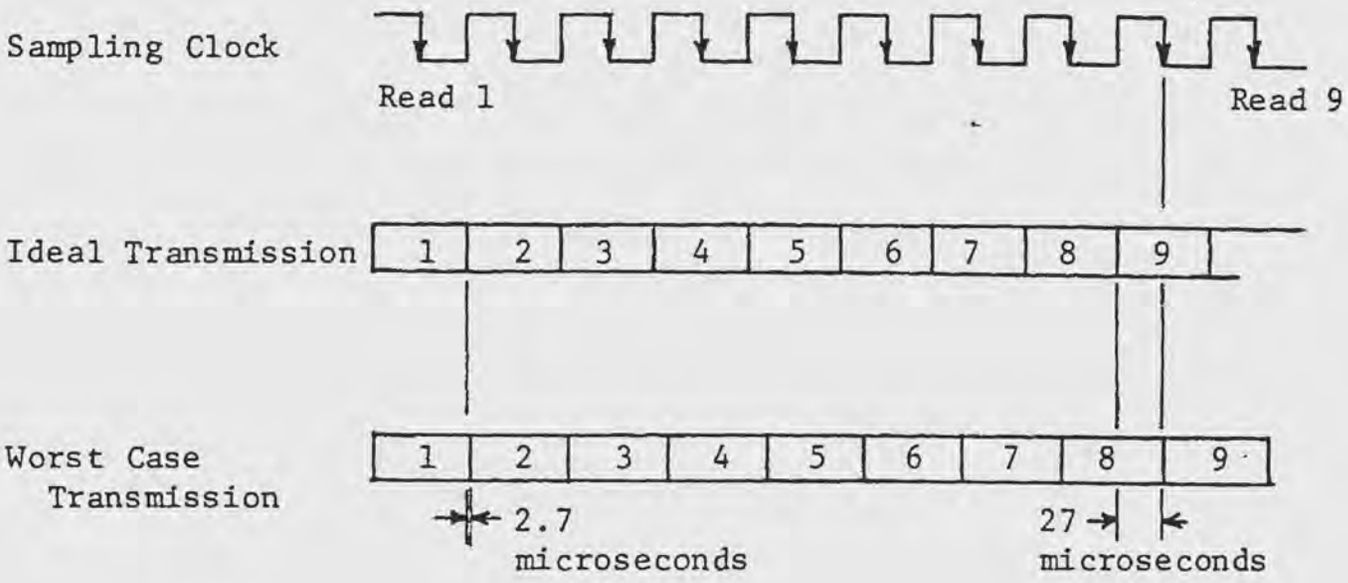


Fig. A3-3. Serial transmission misread

Conversely, if all interrupts were assumed to occur during 2 cycle instructions, the rate mismatch error would act favorably by reducing the final bit error to 22.5 - 4.5,, or 18 microseconds, which would appear to be an acceptable error.

Additional analysis which might prove fruitful in reducing the posibility of error would be a statistical study of the program to determine the relative probability of an interrupt occuring within a two cycle instruction versus its probability of occurence in a one cycle instruction. Once the probabilities were established, the intervals generated by the timer could be set up so that some would assume interrupt occurance during a 1 cycle instruction and some would assume occurance during a 2 cycle instruction.

One final approach to eliminating the transmission error

problem might be to step the input crystal frequency down from its maximum value of 6 megahertz to 5.76 megahertz. As this frequency is an integer multiple of the baud rate, the rate mismatch error is eliminated. However, the error due to the variation in interrupt occurance point becomes slightly larger.

The program steps required to implement the interrupt service routine determine the amount of processing time that will remain available for other processing tasks.

Using the software timing implementation described in section V as a model, it is noted that a minimum of 5 instruction cycles are required to control the character transmission loop. If no additional instructions were required to implement the timer/interrupt scheme, 21-5, or 16 instruction cycles would be available for other processing tasks per bit sent. For the present application, this would allow the processing for several additional inputs to be completed during transmission of the message characters for one. Unfortunately, a considerable number of additional instructions must be executed for the interrupt implementation as shown in the program flow listing below:

1. Jump to interrupt routine - 2 instruction cycles
2. Save accumulator - 1 instruction cycle
3. Load accumulator with character - 1 instruction cycle
   The loop control instructions would be executed at this point
4. Store accumulator into character storage reg. - 1 instruction cycle
5. Stop timer - 1 instruction cycle
6. Load accumulator with timer count - 1 instruction cycle

7. Load timer from accumulator - 1 instruction cycle
8. Start timer - 1 instruction cycle
9. Restore accumulator - 1 instruction cycle
10. Return from interrupt - 2 instruction cycles

These additional steps result in a rather dramatic decrease in the number of instruction cycles available for other processing tasks. Summing the cycle count in the right hand entries, reveals that 12 of the 16 cycles are used. Clearly, under these conditions the timer/interrupt implementation does not appear to be worthwhile.

Some relationship should exist between a processors instruction cycle execution rate, timer count rate, and the desired serial transmission rate which would indicate when it might be cost effective to do the analysis necessary to use a timer/interrupt approach. Additional research on the subject might reveal such a relationship.

LIST OF REFERENCES

1. SBC 80/20 Hardware Reference Manual. 98-317C. Santa Clara, California: Intel Corporation, 1977.

2. ADM-3A Maintainance Manual. Anaheim, California: Lear Sieglar Incorporated.

3. ADM-3A Interactive Display Terminal Operator's Manual. Anaheim, California: Lear Sieglar Incorporated.

4. UPI-41 User's Manual. 980054A. Santa Clara, California: Intel Corporation.

5. TTL Data Book. Dallas, Texas: Texas Instruments Incorporated, 1976.

6. TTL Data Book. Mountain View, California: Fairchild Semiconductor, 1972.

7. Line Driver and Line Receiver Data Book. Dallas, Texas: Texas Instruments Incorporated, 1977.

8. Intel Component Data Catalog. Santa Clara, California: Intel Corporation, 1978.

9. ICE-80 Reference Manual. 98-167B. Santa Clara, California: Intel Corporation, 1975.

10. ICE-80 Operator's Manual. 98-185C. Santa Clara, California: Intel Corporation, 1976.

11. Towle, Herbert C. "Laboratory Record." Orlando, Florida: Naval Training Equipment Center.

12. MCS-48 and UPI-41 Assembly Language Manual. 9800255C. Santa Clara, California: Intel Corporation, 1978.

13. ISIS-II MCS-48/UPI-41 Macro Assembler Version 2.0. Santa Clara, California: Intel Corporation. (Computer Program)

14. MDS-800 Hardware Reference Manual. 980132B. Santa Clara, California: Intel Corporation, 1975.

15. MDS-800 Operator's Manual. 98-129A. Santa Clara, California: Intel Corporation, 1975.

16. Prom Mapper Operator's Manual. 98-236A. Santa Clara, California: Intel Corporation, 1976.

17. ISIS-II Prom Mapper Version 2.0. Santa Clara, California: Intel Corporation. (Computer Program)

18. ISIS-II User's Guide. 9800306D. Santa Clara, California: Intel Corporation.

19. MDS-DOS Hardware Reference Manual. 98-212A. Santa Clara, California: Intel Corporation, 1976.