intel

APPLICATION NOTE

AP-24

August 1977

Application Techniques for the MCS-48™ Family

Lionel Smith
Microcomputer Applications

98-413B

Price $1.00

Related Intel Publications

"MCS-48™ Microcomputer User's Manual"

"Using the 8251 Universal Synchronous/Asynchronous Receiver/Transmitter"

"8255 Programmable Peripheral Interface Applications"

Application Techniques for the MCS-48™ Family

Contents

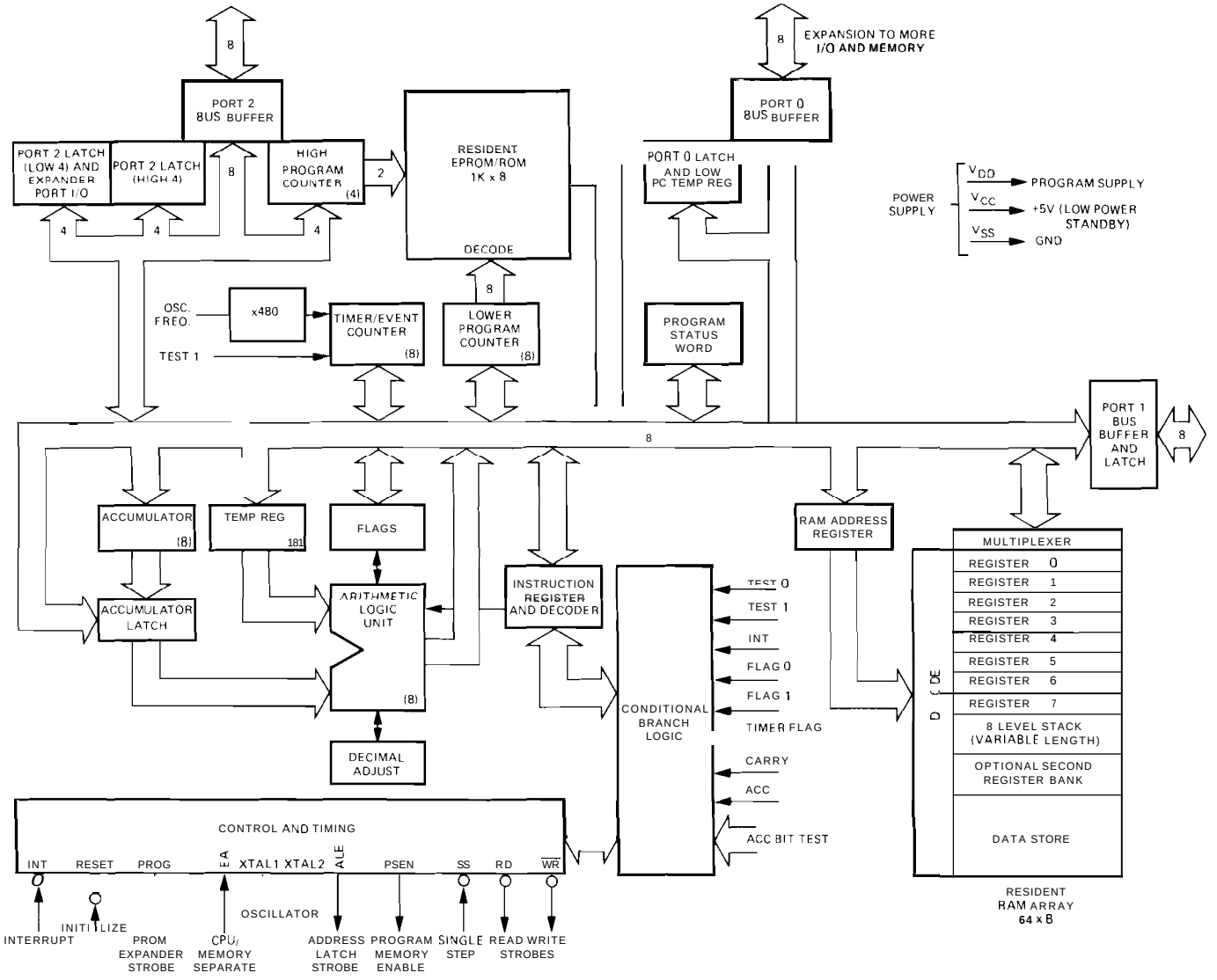
INTRODUCTION

The INTEL® MCS-48™ family consists of a series of seven parts, including three processors, which take advantage of the latest advances in silicon technology to provide the system designer with an effective solution to a wide variety of design problems. The significant contribution of the MCS-48 family is that instead of consisting of integrated microcomputer components it consists of integrated microcomputer systems. A single integrated circuit contains the processor, RAM, ROM (or PROM), a timer. and I/O.

This application note suggests a variety of application techniques which are useful with the MCS-48. Rather than presenting the design of a complete system it describes the implementation of "subsystems" which are common to many microprocessor based systems. The subsystems described are analog input and output, the use of tables for function evaluation, receiving serial code, transmitting serial code, and parity generation. After an overview of the MCS-48 family these areas are discussed in a more or less independent manner.

THE MCS-48™ FAMILY

The processors in the MCS-48 family all share an identical architecture. The only significant difference is the type of on board program storage which is provided. The 8748 (see Figure 1) includes 1024 bytes of erasable, programmable, ROM (EPROM), the 8048 replaces the EPROM with an equivalent amount of mask programmed ROM, and the 8035 provides the CPU function with no on board program storage. All three of these processors



MCS-48™ Internal Structure

INSTRUCTION SET

	Mnemonic	Description	Bytes	Cycle
Accumulator	ADD A,R	Add register to A	1	1
	ADD A. @R	Add data memory to A	1	1
	ADD A, #data	Add immediate to A	2	2
	ADDC A, R	Add register with carry	1	1
	ADDC A, @R	Add data memory with carry	1	1
	ADDC A, #data	Add immediate with carry	2	2
	ANL A, R	And register to A	1	1
	ANL A, @R	And data memory to A	1	1
	ANL A. #data	And immediate to A	2	2
	ORL A, R	Or register to A	1	1
	ORL A, @R	Or data memory to A	1	1
	ORL A, #data	Or immediate to A	2	2
	XRL A, R	Exclusive Or register to A	1	1
	XRL A, @R	Exclusive or data memory to A	1	1
	XRL A. #data	Exclusive or immediate to A	2	2
	INC A	Increment A	1	1
	DEC A	Decrement A	1	1
	CLR A	Clear A	1	1
	CPL A	Complement A	1	1
	DA A	Decimal Adjust A	1	1
	SWAP A	Swap nibbles of A	1	1
	RL A	Rotate A left	1	1
	RLC A	Rotate A left through carry	1	1
	RR A	Rotate A right	1	1
	RRC A	Rotate A right through carry	1	1
Input/Output	IN A,P	Input port to A	1	2
	OUTL P, A	Output A to port	1	2
	ANL P, #data	And immediate to port	2	2
	ORL P, #data	Or immediate to port	2	2
	INS A, BUS	Input BUS to A	1	2
	OUTL BUS, A	Output A to BUS	1	2
	ANL BUS, #data	And immediate to BUS	2	2
	ORL BUS, #data	Or immediate to BUS	2	2
	MOVD A, P	Input Expander port to A	1	2
	MOVD P, A	Output A to Expander port	1	2
	ANLD P, A	And A to Expander port	1	2
	ORLD P, A	Or A to Expander port	1	2
Registers	INC R	Increment register	1	1
	INC @R	Increment data memory	1	1
	DEC R	Decrement register	1	1
Branch	JMP addr	Jump unconditional	2	2
	JMPP @A	Jump indirect	1	2
	DJNZ R, addr	Decrement register and skip	2	2
	JC addr	Jump on Carry = 1	2	2
	JNC addr	Jump on Carry = 0	2	2
	J Z addr	Jump on A Zero	2	2
	JNZ addr	Jump on A not Zero	2	2
	JTO addr	Jump on TO = 1	2	2
	JNTO addr	Jump on TO = 0	2	2
	JT1 addr	Jump on T1 = 1	2	2
	JNT1 addr	Jump on T1 = 0	2	2
	JFO addr	Jump on FO = 1	2	2
	JF1 addr	Jump on F1 = 1	2	2
	JTF addr	Jump on timer flag	2	2
	JNI addr	Jump on $\overline{INT}$ = 0	2	2
	JBb addr	Jump on Accumulator Bit	2	2

	Mnemonic	Description	Bytes	Cycles
Subroutine	CALL	Jump to subroutine	2	2
	RET	Return	1	2
	RETR	Return and restore status	1	2
Flags	CLR C	Clear Carry	1	1
	CPL C	Complement Carry	1	1
	CLR FO	Clear Flag 0	1	1
	CPL FO	Complement Flag 0	1	1
	CLR F1	Clear Flag 1	1	1
	CPL F1	Complement Flag 1	1	1
Data Mov	MOV A, R	Move register to A	1	1
	MOV A, @R	Move data memory to A	1	1
	MOV A, #data	Move immediate to A	2	2
	MOV R, A	Move A to register	1	1
	MOV @R,A	Move A to data memory	1	1
	MOV R, #data	Move immediate to register	2	2
	MOV @R, #data	Move immediate to data memory	2	2
	MOV A, PSW	Move PSW to A	1	1
	MOV PSW, A	Move A to PSW	1	1
	XCH A, R	Exchange A and register	1	1
	XCH A, @R	Exchange A and data memory	1	1
	XCHD A, @R	Exchange nibble of A and register	1	1
	MOVX A, @R	Move external data memory to A	1	2
	MOVX @R, A	Move A to external data memory	1	2
	MOVP A, @A	Move to A from current page	1	2
	MOVP3 A, @A	Move to A from Page 3	1	2
Tir/Count	MOV A, T	Read Timer/Counter	1	1
	MOV T, A	Load Timer/Counter	1	1
	STRTT	Start Timer	1	1
	STRTCNT	Start Counter	1	1
	STOP TCNT	Stop TimerlCounter	1	1
	EN TCNTI	Enable Timer/Counter Interrupt	1	1
	DIS TCNTI	Disable Timer/Counter Interrupt	1	1
Cont ol	EN I	Enable external interrupt	1	1
	DIS I	Disable external interrupt	1	1
	SELRBO	Select register bank 0	1	1
	SEL RB1	Select register bank 1	1	1
	SEL MBO	Select memory bank 0	1	1
	SEL MB1	Select memory bank 1	1	1
	ENTO CLK	Enable Clock output on TO	1	1
	NOP	No Operation	1	1

Figure 2. 8048/8748/8035 Instruction Set

operate from a single 5-volt power supply. The 8748 requires an additional 25-volt supply only while the on board EPROM is being programmed. When installed in a system only the 5-volt supply is needed. Aside from program storage, these chips include 64 bytes of data storage (RAM), an eight bit timer which can also be used to count external events, 27 programmable I/O pins and the processor itself. The processor offers a wide range of instruction capability including rnany designed for bit, nibble, and byte manipulation. The instruction set is summarized in Figure 2.

Aside from the processors, the MCS-48 family includes 4 devices: one pure I/O device and 3 combination memory and I/O devices. The pure I/O device is the 8243, a device which is connected to a special 4 bit bus provided by the MCS-48 processors and which provides 16 I/O pins which can be programmatically controlled.

The combination memory and I/O devices consist of the 8355, the 8755, and the 8155. The 8355 and the 8755 both provide 2,048 bytes of program storage and two eight bit data ports. The only difference between these devices is that the 8355 contains masked program ROM and the 8755 contains EPROM. The 8155 combines 256 bytes of data storage (RAM), two eight bit data ports, a six bit control port, and a 14 bit prograrnrnable timer.

Figure 3 shows the various system configurations which can be achieved using the MCS-48 family of parts. It should also be noted that eight of the processors' I/O lines have been configured as a bidirectional bus which can be used to interface to standard Intel peripheral parts such as the 8251 USART (for serial I/O), the 8255A PPI (provides 24 I/O lines) and the complete range of memory components.

More detailed information concerning the MCS-48 farnily can be obtained from the "MCS-48 Microcomputer User's Manual" which provides a complete description of the MCS-48 family and its members. A general familiarity with this document will make the application techniques which follow easier to understand.

ANALOG I/O

If analog I/O is required for a MCS-48™ system there are many alternatives available from the makers of analog I/O modules. By searching through their catalogs it is possible to find almost any cornbination of features which is technically feasible. Perhaps the best example of such modules are the MP-10 and MP-20 hybrid modules recently introduced by Burr-Brown Research Corporation. The MP-10 provides two analog outputs and the MP-20 provides 16 analog inputs. Both of these units were specifically designed to interface with microprocessors.

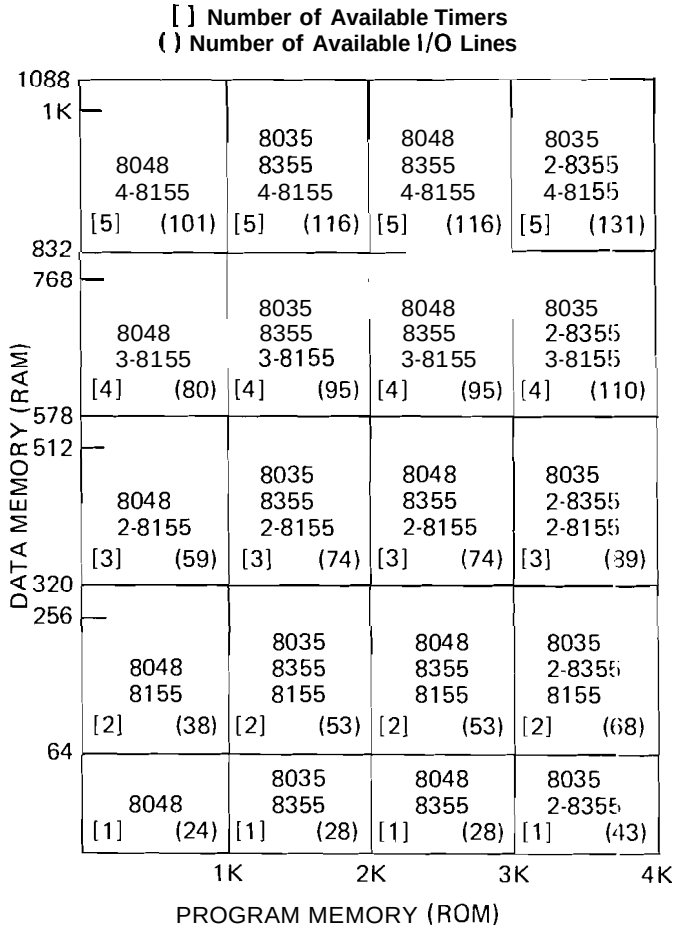


Figure 3. The Expanded MCS-48™ System

A block diagram of the MP-10 is shown in Figure 4. It consists of two eight bit digital to analog conferters, two eight bit latches which are loaded from the data bus, and address decoding logic to deterrnine when the latches should be loaded. The D/A converters each generate an analog output in the range of 10 volts with an output impedance of Ω. Accuracy is ±0.4% of full scale and the output is stable 25μsec after the eight bit binary data is loaded into the appropriate latch. The latches are loaded by the write pulse ($\overline{WR}$) whenever the proper address is presented to the MP-10. The lower two addresses (A_0 and A_1) are used in-ernally by the device. Addresses A2 & A3 are compared with the address determination inputs B_2 and B3. If their signals are found to be equal, and if addresses A4-A13 are all high, then the device is selected and one of the latches will be loadzd. Address bit A_1 selects between output 1 and output 2. If address bit A_0 is set then the initialization channel of the DIA is selected. In order to prepare for operation a data pattern of 80_H must

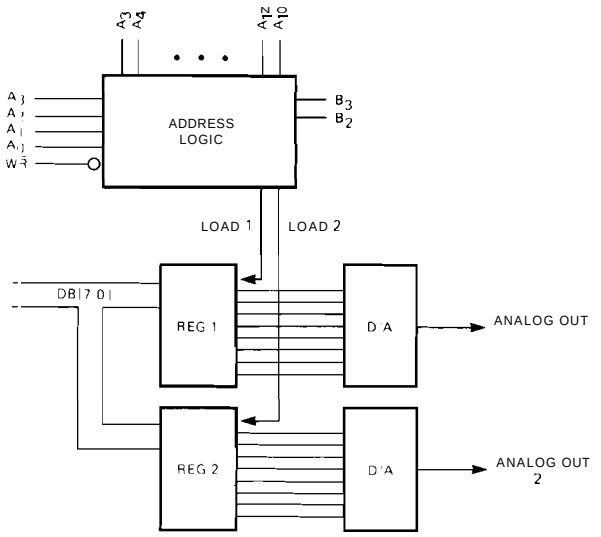


Figure 4. MP-10 Block Diagram

be output to this channel following the reset of the device.

A block diagram of the MP-20 analog to digital converter is shown in figure 5. This unit consists of a 16 input analog multiplexer, an instrumentation amplifier, an eight bit successive approximation analog to digital converter, and control logic. The 16 input multiplexer can be used to input either 16 single ended or 8 differential inputs. The output from the multiplexer is fed into the instrumentation amplifier which is configured so that it can easily be strapped for single ended 0-5 volt inputs, single ended ±5 volt inputs, or differential 0-5 volt signals. Provisions are made for an external gain control resistor on the amplifier. The gain control equation is:

$$G = 2 + \frac{50k\Omega}{R_{ext}}$$

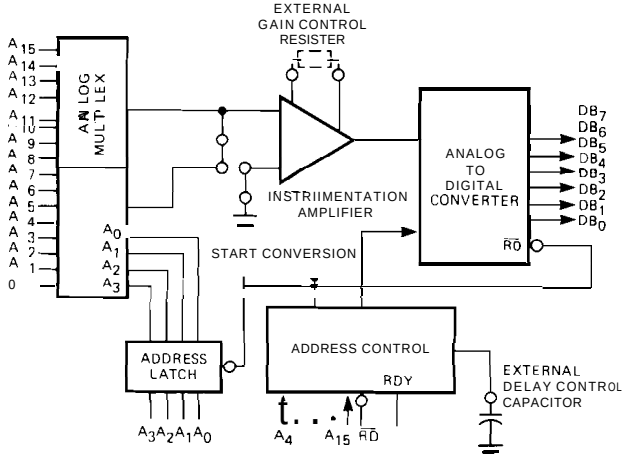


Figure 5. MP-20 Analog Subsystem

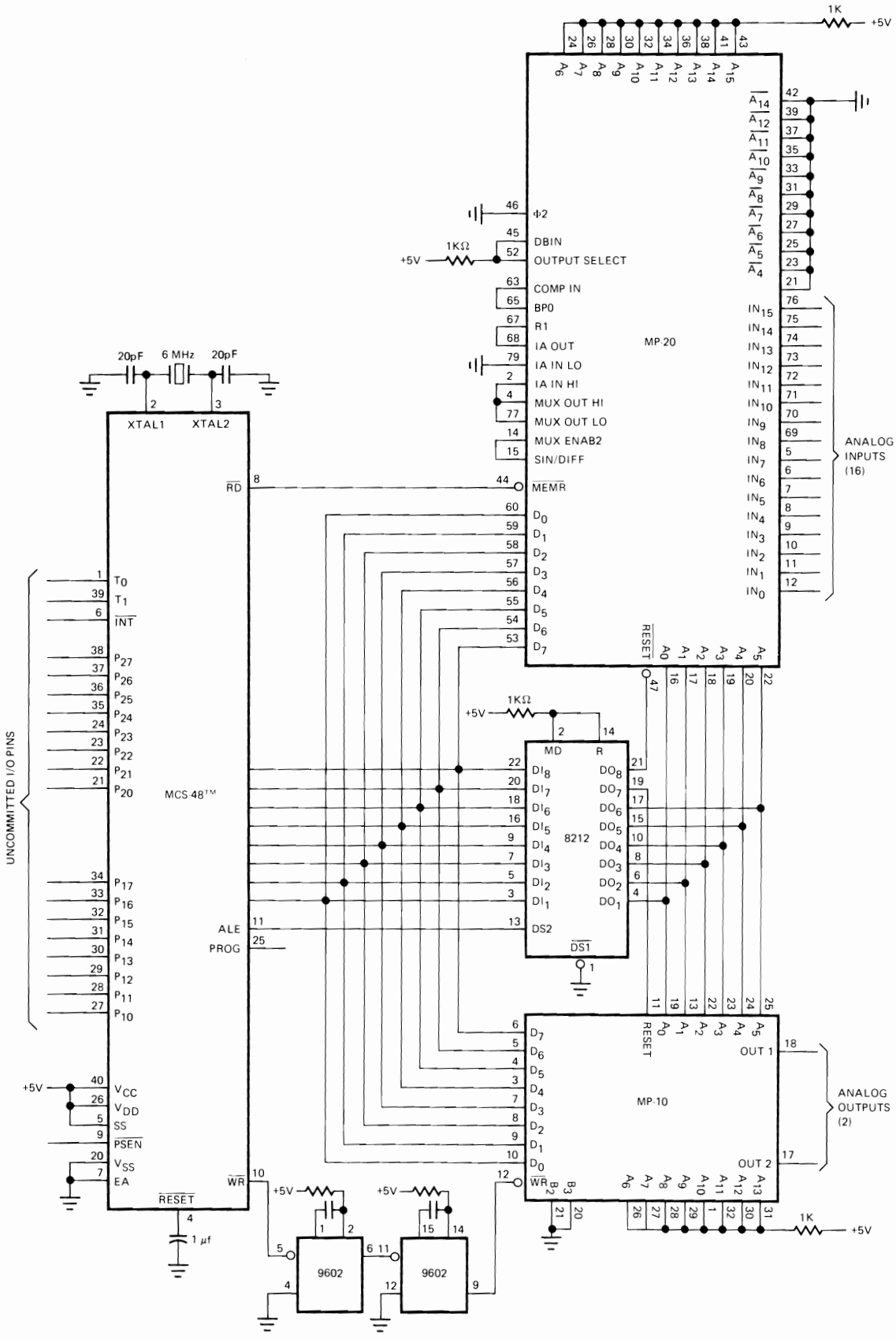
With no R_{ext} ($R_{ext} = \infty$) the gain is two and the input is 0-5 or ±5 volts full scale. Adding an external resistor results in higher gain so that low level (±50mV) signals from thermocouples and strain gauges can be accommodated. The output from the amplifier is applied to the actual A/D converter which provides an eight bit output with guaranteed monotonicity and an accuracy of ±0.4% of full scale. Note that this accuracy is specified for the entire module, not just for the converter itself. The control logic monitors address lines A_{15} through A_4 to determine when the address of the unit has been selected. An address that the unit will respond to is determined by 11 address control pins, labeled $\overline{A4}$ through $\overline{A14}$. If one of these pins is tied to a logic 0 then the corresponding address pin must be high in order for the unit to be selected. If the pin is tied to a logic 1 then the corresponding address pin must be low. If the address of the module is selected when $\overline{MEMR}$ pulse occurs, the lower four addresses (A_3-A_0) are stored in a latch which addresses the multiplexer. The coincidence of the proper address and $\overline{MEMR}$ also initiates a conversion and gates the output of the converter on to the eight bit data bus.

The control logic of the MP-20 was designed to operate directly with an MCS-80™ system. When a $\overline{MEMR}$ occurs and a conversion is initiated the MP-20 generates a READY signal which is used to extend the cycle of the 8080A for the duration of the conversion. READY is brought high after the conversion is complete which allows the 8080A to initiate a conversion and read the resulting data in a single, albeit long, memory or I/O cycle. The conversion time of the MP-20 depends on the gain selected for the amplifier. With no external resistor ($R = \infty$) the gain is two and the conversion time is 35 psec. For R = 510Ω the gain is:

$$G = 2 + \frac{50k\Omega}{.51k\Omega} \cong 100$$

and the conversion time becomes 100μsec. These settling times are specified in the MP-20 data sheet and range from 35 to 175 microseconds. The READY timing is controlled by an external capacitor. For a gain of 2 no external capacitor is required but if higher gains are selected a capacitor is needed to extend the timing.

A schematic showing both the MP-10 D/A and the NIP-20 A/D connected to the 8748 is shown in Figure 6. This configuration, which consists of only four major components, gives an excellent example of what modern technology can do for



MCS-48™ Based Analog Processor

the system designer. The four components provide:

a. An eight bit microprocessor
b. 64 bytes of RAM
c. 1024 bytes of UV erasable PROM
d. A timer/event counter
e. 16 digital I/O pins
f. 2 testable input pins
g. An interrupt capability
h. 16 eight bit analog inputs
i. 2 eight bit analog outputs

The MCS-48 communicates with the D/A and A/D converters in a memory mapped mode (i.e., it treats the devices as if they were external RAM). By setting an address in either R_0 or R_1 and then executing a MOVX the software can transfer data between the accumulator and the analog I/O. When the MCS-48 executes the MOVX instruction it first sends the eight bit address out on the bus and strobes it into the 8212 latch with the ALE (Address Latch Enable) signal. After the address is latched, the MCS-48 uses the same bus to transfer data to or from the accumulator. If data is being sent out (MOVX @Rj, A) the $\overline{WR}$ strobe is used; if the data is being moved into the accumulator (MOVX A, @Rj) the $\overline{RD}$ strobe is used. The one shots on the $\overline{WR}$ line are used to delay the write strobe of the MCS-48 to meet the data set up specifications of the MP-10.

In order to provide reset capability for the analog devices without dedicating an I/O pin from the MCS-48, special addresses are used as reset channels. Executing any MOVX with an address of 0XXXXXXX will reset the A/D module; a similar operation with an address of X1XXXXXX will reset the D/A; a MOVX with an address of 01XXXXXX will reset both devices. All data transfers are accomplished with the upper two bits of the address field equal to 10. A summary of the addressing of the analog devices is shown in Table 1. Notice that except for an initialization channel for the D/A (which must be written to following a reset to initialize its internal logic) all channels involve some form of data transfer.

Table 1. Analog Interface Addresses

INPUT OR OUTPUT	
0 X X X X X X X X	Reset A/D
X 1 X X X X X X	Reset D/A
INPUT	
0 0 1 1 n n n n	Read A/D Channel n n n n
OUTPUT	
1 0 1 1 0 0 0 1	Initialize D/A
1 0 1 1 0 0 0 0	Write Channel 1
1 0 1 1 0 0 1 0	Write Channel 2

As was mentioned previously, the MP-20 was designed to use the READY line of the 8080A. Obviously this presents a problem since the MCS-48 does not support a READY line (with its attendant requirement of entering WAIT state). The necessity of a READY input can be overcome by performing a read operation to set the channel address, waiting the required delay (35 μsec for a gain of two) and then performing a second read to actually obtain the data. The second read will read in the data from the channel selected by the first read irrespective of the channel selected for the second read. Thus it is possible to use the second read to set up the channel for the third read. Each read can read in the current channel and select the next channel for conversion.

The MP-20 is shown in Figure 6 strapped to input 16 single ended ±5 volts signals. Programs which were used to test this configuration are shown in Figure 7. The first of these programs uses the D/A converter to generate sawtooth waveforms by outputting an incrementing value to the D/A converters. The second program scans the analog inputs and stores their digital values in a table located in RAM.

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                    0
                    1
                    2 ; ------------------------------
                    3 ; TEST PROGRAM FOR ANALOG OUTPUT
                    4 ;   THIS PROGRAM OUTPUTS A SAW-
                    5 ;   TOOTH WAVEFORM BY OUTPUTING
                    6 ;   AN INCREMENTING PATTRERN.
                    7 ; ------------------------------
                    8
                    9 ; -------
                   10 ; EQUATES
                   11 ; -------
                   12
00B3               13 INITCH  EQU    0B3H    ; D/A INITIALIZATION CHANNEL
0080               14 INITDT  EQU    80H     ; D/A INITIALIZATION DATA
00B0               15 DATCH   EQU    0B0H    ; D/A DATA CHANNEL
                   16
                   17 ; -------------
                   18 ; START OF TEST
                   19 ; -------------
0100               20         ORG    100H
                   21                        ; INITIALIZE D/A
0100 2380          22 START:  MOV    A,#INITDT
0102 B8B3          23         MOV    R0,#INITCH
0104 90            24         MOVX   @R0,A
                   25                        ; TEST LOOP-OUTPUT SAWTOOTH
0105 B8B0          26 LOOP:   MOV    R0,#DATCH
0107 17            27         INC    A
0108 90            28         MOVX   @R0,A
0109 2405          29         JMP    LOOP
                   30                        ; END OF PROGRAM
                   31         END
```

Figure 7a. D/A Exercise Program

```
LOC  OBJ        SEQ     SOURCE STATEMENT

                  0
                  1
                  2 ; ---------------------------------------
                  3 ; TEST PROGRAM FOR ANALOG INPUT
                  4 ;   THIS PROGRAM SCANS THE INPUT CHANNELS
                  5 ;   AND STORES THE READINGS IN A TABLE
                  6 ;   STARTING AT BUFF.
                  7 ; ---------------------------------------
                  8
                  9 ; -------
                 10 ; EQUATES
                 11 ; -------
                 12
0020             13 BUFF    EQU     20H     ; START OF BUFFER
000F             14 MAXCH   EQU     15      ; NO OF ANALOG INPUTS
00B0             15 AINCH   EQU     0B0H    ; BASE ADDRESS OF ANALOG INPUTS
0005             16 TICK    EQU     5       ; EXECUTION TIME OF DJNZ
                 17
                 18 ; -------------
                 19 ; START OF TEST
                 20 ; -------------
0100             21         ORG     100H
                 22                         ; SETUP TO SCAN ANALOG INPUTS
0100 B92F        23 START:  MOV     R1,#BUFF+MAXCH
0102 BB0F        24         MOV     R3,#MAXCH
0104 B8BF        25         MOV     R0,#(AINCH+MAXCH)
                 26                         ; SELECT CHANNEL 15
0106 80          27         MOVX    A,@R0
                 28                         ; WAIT >40 MICROSECONDS
0107 BC08        29         MOV     R4,#40/TICK
0109 EC09        30         DJNZ    R4,$
                 31                         ; NOW SCAN ANALOGS
010B 18          32 LOOP:   DEC     R0
                 33                         ; GET DATA
010C 80          34         MOVX    A,@R0
                 35                         ; MOVE INTO BUFFER
010D A1          36         MOV     @R1,A
                 37                         ; DECREMENT BUFFER POINT
010E C9          38         DEC     R1
                 39                         ; PAD 20 MICROSEC
010F BC04        40         MOV     R4,#20/TICK
0111 1111        41         DJNZ    R4,$
                 42                         ; LOOP UNTIL DONE
0113 EB0B        43         DJNZ    R3,LOOP
                 44                         ; REPEAT TEST FOREVER
0115 2400        45         JMP     START
                 46                         ; END OF PROGRAM
                 47         END
```

Figure 7b. A/D Exercise Program

TABLE LOOKUP TECHNIQUES

In the previous section the interface between analog I/O devices and the MCS-48™ was discussed. In many applications involving analog I/O one quickly finds that nature is inherently nonlinear, and the mathematics involved in 'linearizing it' can tax the computational power of the microprocessor, particularly if it has other tasks to perform. Problems of this nature are good candidates for the use of tables.

As an example of how tables can be used as part of an analog output scheme, consider a system which requires an MCS-48 to output a variable frequency sinusoidal waveform. One method of performing this function would be to use the timer to generate an interrupt at a fixed rate of *256* times the desired output frequency. At each interrupt the appropriate value of the sine function could be calculated from the MacLaurin series:

$$\text{Sin } x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \cdots \frac{(-1)^k \, x \, 2k+1}{(2K+1)!}$$

Where K is chosen to be large enough to provide the required accuracy.

The above calculation, although conceptually simple, would be time consuming and would severely limit the possible output frequencies which could be obtained. As an alternative to calculating these values in real time, the values could be precalculated off line and stored in a table. Upon each interrupt the MCS-48 would merely have to retrieve the appropriate value from the table and output it to the D/A converter. the MCS-48 provides a special instruction which can be used to access data in a table. If the table is stored in the last *256* bytes of the first kilobyte of MCS-48 memory then the table lookup can be performed by loading the independent variable (time in this case) into the accumulator and executing the instruction.

MOVP3 A, @ A

This instruction uses the initial contents of the accumulator to index into page **3** of program storage. The location pointed to is read and the contents placed in the accumulator. If (as is often the case) a table of fewer than *256* entries is required, then the table can be located in any page of program memory and the instruction:

MOVP A, @ A

can be used to retrieve data from the table. This instruction operates in the same manner as does the previous instruction except that the current page of program storage is assumed to ccntain the table.

If it is possible to devote slightly more of the microprocessor's time to the table look up process, then a much smaller table can often be utilizzd by taking advantage of interpolation to determine values of the function between values which are actual entries in the table. As an example of this

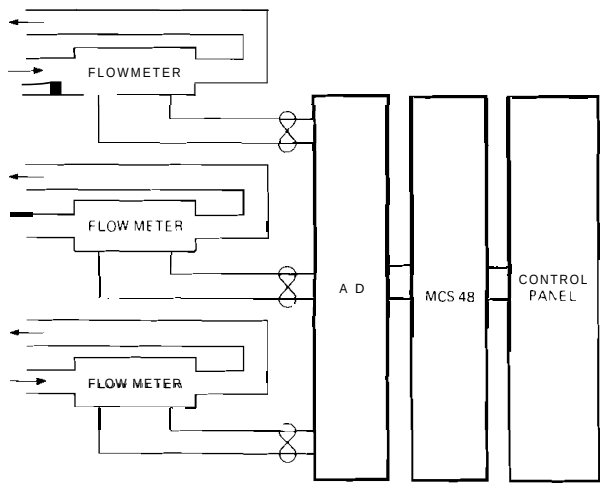


Figure 8. Flow Monitoring System

process consider the hypothetical system shown in Figure 8. The purpose of this system is to measure the flow through the three pipes, add them, and display the total flow on the control panel. The system consists of three flow meters which generate a differential voltage which is some function of flow, an A/D system with at least three differential inputs, an MCS-48, and a control panel. The schematic shown in Figure 6 could easily become part of this system, with the spare digital I/O of the MCS-48 used as an interface to the control panel. The simplicity of this system is clouded by the flow transducers, which are assumed to be not only nonlinear but also to require individual calibration (this is not an unreasonable assumption for a flow transducer). By using a table look up process and an 8748 the flow transducers can be calibrated and the results of the calibration tests stored directly in tables in the 8748. (The 8748 has a PROM in place of the ROM of the 8048 and thus makes such 'one off' programming practical.)

The results which might be obtained from calibrating one of the flow meters is shown in Figure 9. The results are plotted as gals/hour versus the measured voltage generated by the transducer. The voltage is shown in hexadecimal form so that it corresponds directly to the digital output of the analog to digital converter. The flow required to generate seventeen evenly spaced voltages (OH-1OOH in steps of 10H) has been measured and plotted. This information is shown in tabular form in Figure 10. It is necessary to generate a program which will convert any measured input from OOH to FFH into the flow in units which can be interpreted by a human operator. This can easily be done by simple interpolation.

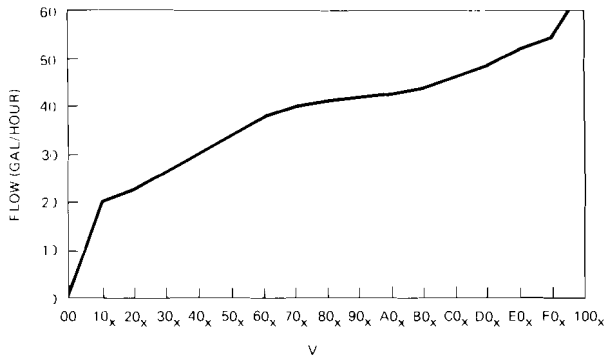


Figure 9. Flow Calibration Curve

TRANSDUCER VOLTAGE (HEX)	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100
MEASURED FLOW (GAL HOUR)	0	10	22	26	30	34	38	40	41	42 (43	45	48	49	53	56	63	

Figure 10. Tabulated Flow Data

The eight bits of independent variable (voltage) can be looked on as two four bit fields. The most significant four bits (7-4) will be used to retrieve one of the table values. The lower four bits (3-0) will be used to interpolate between this value and the value retrieved from the next higher location in the table. If the upper four bits are given the symbol I and the lower four bits the symbol N, then the interpolation can be expressed as:

$$F(x) = F(I) + \frac{N}{16} [F(I+1) - F(I)]$$

Where x is the measured voltage and F(x) is the corresponding flow.

If, as an example, the transducer voltage was measured as 48H then the flow (ref. Figure 10) would be:

$$F = 30 + \frac{8}{16} (34-30) = 32$$

A subroutine which implements this calculation is shown in Figure 11. Before it is called the independent variable (V) is placed in the accumulator and register R1 is set to point at the first value in the table. Aside from simple additions and subtractions the only arithmetic required is to multiply two values and then divide them by 16. The multiplication is handled via a subroutine which is also shown in Figure 11. The division by 16 can be performed by a four place right shift followed by a rounding operation. The routine shown will handle a monotonic increasing function of a single independent variable. Fairly simple modifications are required for nonmonotonic functions. Functions of two variables can be handled by interpolating on a plane rather than along a straight line. Although this is more time consuming, requiring an interpolation for each of the independent variables and a third to interpolate the final answer, it still provides a simple means of quickly calculating the required function. The use of tables can offer a powerful technique for function evaluation to the designer.

RECEIVING SERIAL CODE—BASIC APPROACHES

Many microprocessor based systems require some form of serial communication. Serial communication is extensively used because it allows two or more pieces of equipment to exchange information with a minimal number of interconnecting wires. The minimization of interconnecting wires results in simpler, cheaper, interconnects because fewer (or smaller) cables and connectors are required. Since the required number of drivers and receivers required is reduced, it can become economically feasible to provide much higher noise immunity

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                  0 ; ****************************************
                  1 ;
                  2 ;       APPROX
                  3 ;         AI ENTRY RI POINTSAT TABLE
                  4 ;                 A HAS INDEPENDANT VARIABLE
                  5 ;
                  6 ; ****************************************

                  8 . --------
                  9 ; EQUATES
                 10 . -------
                 11
0000             12 RX0     EQU     R0      ; POINTER 0
0001             13 RX1     EQU     R1      ; POINTER 1
0002             14 AEX     EQU     R2      ; EXTENSION OF A REGISTER
0003             15 COUNT   EQU     R3      ; COUNTER
0004             16 TEMP    EQU     R4      ; TEMP STORAGE
                 17
                 18 . ------------
                 19 ; APPROXIMATION
                 20 . ------------
                 21
0100             22         ORG     100H
                 23                         ; POINT RX0 AT TEMP
0100 B804        24 APPROX: MOV     RX0,#TEMP
                 25                         ; TEMP=N AND 0FH
                 26                         ; A=P AND 0FH
0102 B000        27         MOV     @RX0,#0
0104 36          28         XCHD    A,@RX0
0105 47          29         SWAP    A
                 30                         ; RX1=BASE+A
0106 69          31         ADD     A,RX1
0107 A9          32         MOV     RX1,A
                 33                         ; RX1=TABLE(P)
                 34                         ; A=TABLE(P+1)
0108 E3          35         MOVP3   A,@A
0109 29          36         XCH     A,RX1
010A 17          37         INC     A
010B E3          38         MOVP3   A,@A
                 39                         ; A=TABLE(P+1)-TABLE(P)
010C 37          40         CPL     A
010D 69          41         ADD     A,RX1
010E 37          42         CPL     A
                 43                         ; A=N*A/16
010F 341D        44         CALL    MULT
0111 B502        45         MOV     RX0,#AEX
0113 30          46         XCHD    A,@RX0
0114 47          47         SWAP    A
0115 2A          48         XCh     A,AEX
0116 7219        49         JB3     ADJUST
0118 2A          50         XCh     A,AEX
0119 2A          51 ADJUST: XCH     A,AEX
011A 17          52         INC     A
                 53                         ; A=A+TABLE(P)
011B 69          54         ADD     A,RX1
                 55                         ; RETURN

LOC  OBJ        SEQ        SOURCE STATEMENT

011C 83          56         RET
                 57
                 58
                 59 . --------
                 60 ; MULTIPLY
                 61 . --------
                 62                         ; SET UP CWNT AND AEX
011D B608        63 MULT:   MOV     COUNT,#8
011F BA00        64         MOV     AEX,#0
                 65                         ; CLEAR CARRY
0121 97          66 LOOPA:  CLR     C
                 67                         ; IF MULTIPLIER[0] <> I THEN SHIFT PRODUCT
0122 1228        68 LOOPB:  JB0     SSUM
0124 2A          69         XCH     A,AEX
0125 67          70         RRC     A
0126 2A          71         XCH     A,AEX
0127 67          72         RRC     A
                 73                         ; LOOP UNTIL DONE
0128 E822        74         DJNZ    COUNT,LOOPB
012A 83          75         RET
                 76                         ; ELSE ADD MULTIPLIER AND SHIFT PRODUCT
012B 2A          77 SSUM:   XCH     A,AEX
012C 60          78         ADD     A,@RX0
012D 67          79         RRC     A
012E 2A          80         XCH     A,AEX
012F 67          81         RRC     A
                 82                         ; LOOP UNTIL DONE
0130 E821        83         DJNZ    COUNT,LOOPA
0132 83          84         RET
                 85
                 86
                 87 . --------------------
                 88 ; TABLE TO TEST PROGRAM
                 89 . --------------------
                 90
0380             91         ORG     380H
                 92
0380 00          93 TABLE:  DB      00      ; THIS TABLE IS FROM FIG 10
0381 0A          94         DB      10
0382 16          95         DB      22
0383 1A          96         DB      26
0384 1E          97         DB      30
0385 22          98         DB      34
0386 26          99         DB      38
0387 28         100         DB      40
0388 29        1111         DB      41
038A 2A         102         DB      42
038A 2B         103         DB      43
038B 2D         104         DB      45
038C 30         105         DB      48
038D 31        1U6          DB      49
038E 35         187         DB      53
038F 38         168         DB      56
0390 3F         109         DB      63
                110
                111         END
```

Figure 11. Table Lookup With Interpolation

with more sophisticated (and expensive) line terminators. The final, and usually most persuasive, argument in favor of serial cornrnunication is that it may be the only method available to accomplish the job. The obvious example of this is telecommunications where it is necessary to encode parallel information into serial format in order to communicate via the telephone network. The intent of this section is to show how the facilities of the MCS-48™ can be brought to bear on the problem of serial communication.

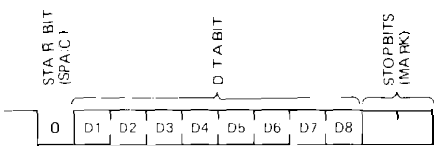


Figure 12. Serial ASCII Code

Probably the most common form of serial communication is that used by the obiquitous Teletype-serial ASCII. This format, shown in Figure 12, consists of a START bit (0 or SPACE) followed by eight data bits which are in turn followed by two STOP bits (1 or MARK). In actual practice the eighth data bit usually consists of even parity on the remaining seven data bits; for the purposes of this discussion the eighth bit will be considered only as data. A minor variation of this format deletes one of the STOP bits. An algorithm which might be used to sample serial data under software control using a microprocessor is shown in Figure 13. Th: basic intent of this algorithm is to minimize the effects of distortion and transmission rate variations on the reliability of the communication by sampling each data bit as close to its center as possible. Upon entry to this routine the software first samples the incorning data in a tight loop until it is sensed as a MARK (logical one). As soon as a MARK is detected, a second loop is entered during which the software waits until the received data goes to a SPACE (logical zero). The purpose of this construction is to detect as accurately as possible the leading edge of the START bit. This instant of time will be used as a reference point for sampling all of the following bits in the character. After sensing the leading edge of the START bit a wait of one half the expected bit time is implemented. The period of the incoming signal is called P for convenience. At the end of this wail the serial line is tested—if it is MARK then the START bit was

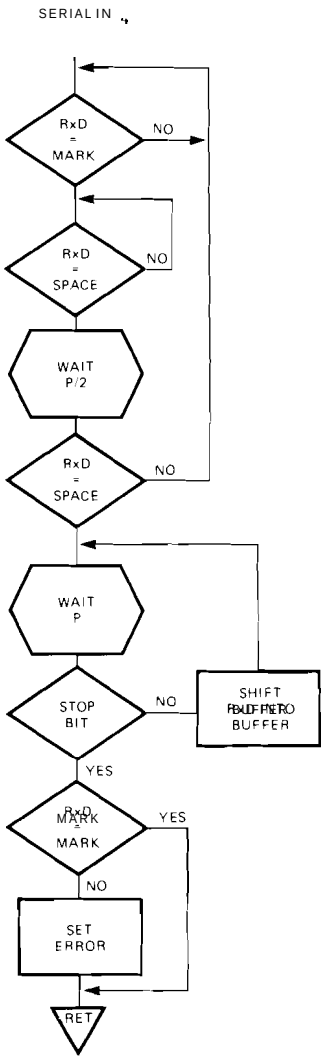


Figure 13. Sample Serial Input Routine

invalid and the process is reinitialized. If the line is still a SPACE, then the START bit is assumed to be valid and a delay of one bit time is started. At the completion of the delay the first data bit is sampled and a new delay of one bit time is initiated. This process is repeated until all eight data bits have been sampled. The last bit sampled is checked to determine if it is a valid STOP bit (a MARK). If it is, the character is assumed to be valid; if it is not, the character has a framing error and is probably invalid. A listing of a program which implements the above procedure is shown in Figure 14.

A disadvantage of the approach outlined in Figure 13 is that while the processor is inputting data serially it must totally dedicate itself to this task. Accurate timing can only be maintained if the program remains in a tight wait loop without allowing itself to be diverted to other functions. During reception of a character from a Teletype the processor will spend only a 100μsecs or so processing data and the rest of the 100 millisecs waiting to do the processing at the right time. This lack of efficiency (approximately 0.1%) in the utilization of processing power is why devices such as the 8251 USART find broad application in microprocessor systems.

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                    0 ; ********************************
                    1 ;
                    2 ;       SIMPLE SERIAL INPUT
                    3 ;       -THIS CODE ASSUMES RXD IS
                    4 ;        CONNECTED TO PIN T0
                    5 ; ********************************
                    7
                    8 ; -------
                    9 ; EQUATES
                   10 ; -------
                   11
0002               12 COUNT    EQU     R2        ; COUNTER
0008               13 BITNO    EQU     8         ; NO OF BITS TO RECEIVE
111112             14 DLYHI    EQU     2         ; HI DLY COUNT
00A4               15 DLYLO    EQU     0A4H      ; LO DLY COUNT
                   16
0100               17          ORG     100H
                   18                            ; LOOP UNTIL RXD=MARK
0100 26011         19 SERIN:   JNT0    $
                   20                            ; NOW LOOP UNTIL RXD=SPACE
0102 3602          21          JT0
                   22                            ; WAIT 1/2 BIT TIME
0104 341C          23          CALL    HBIT
                   24                            ; IF FALSE START REINTIALIZE
01116 3600         25          JT0     SERIN
                   26                            ; ELSE SET BIT COUNT
0108 BA09          27          MOV     COUNT,#BITNO+1
                   28                            ; WAIT 1 BIT TIME
010A 341C          29 LOOP:    CALL    HBIT
010C 341C          30          CALL    HBIT
                   31                            ; DECREMENT COUNT
                   32                            ; -IF ZERO EXIT WITH CARRY SET ON
                   33                            ; -FRAMING ERROR
010E EA15          34          WNZ     COUNT,LOAD
0110 97            35          CLR     C
0111 3614          36          JT0     EXIT
0113 A7            37          CPL     C
0114 83            38 EXIT:    RET
                   39                            ; LOAD DATA
0115 97            40 LOAD:    CLR     C
0116 2619          41          JNT0    LLLA
0118 A7            42          CPL     C
0119 67            43 LLLA:    RRC     A
                   44                            ; AND LOOP
011A 240A          45          J W     LOOP
                   46
                   47 ; ----------------------
                   48 ; DELAY ONE HALF BIT TIME
                   49 ; ----------------------
                   50
                   51                            ; SET UP LOOP
011C BC02          52 HBIT:    MOV     R4,#DLYHI
                   53                            ; LOOP M I L TIME DONE
011E BBA4          54 HLOOP:   MOV     R3,#DLYLO
0120 EB20          55          WNZ     R3,$
0122 EC1E          56          DJNZ    R4,HLOOP
0124 83            57          RET
                   58                            ; END OF PROGRAM
                   59          END
```

Figure 14. Simple Serial Input

The 8251 USART is simple to interface to the MSC-48. Figure 15 shows such an interface. The USART requires a high speed clock (CLK), an initilization signal (RESET), data clocks (TxC and RxC), and data in order to operate. A circuit showing the connection of an 8748 to an 8251 USART is shown in Figure 15. In the circuit shown the high speed clock (which is used for internal sequencing by the USART) is provided by con-

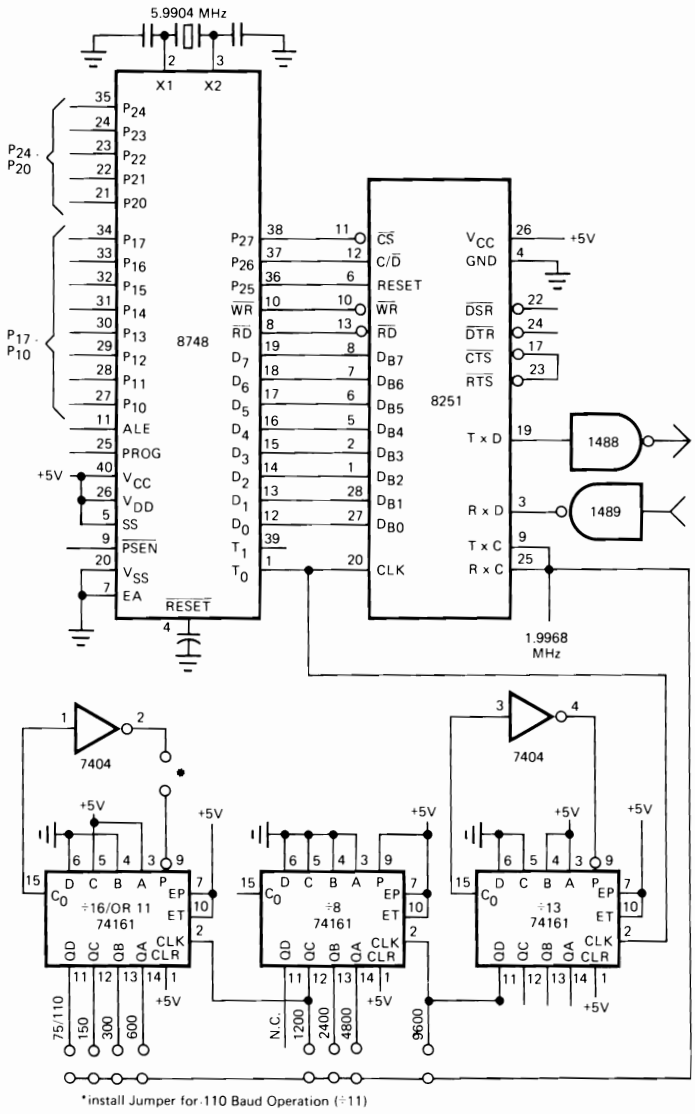


Figure 15. MCS-48™ to 8251 Interface

necting the CLK signal of the USART to the T_0 pin of the MCS-48. The T_0 pin of the MCS-48 can either be used as a directly testable input pin or it can become, under program control, an output pin which oscillates at one third of the crystal frequency. (Note that once this pin is designated by the software to be an output it will remain so until the system is reset.) In Figure 15 the crystal frequency is 5.9904 MHz so the clock provided to the 8251 is 1.9968 MHz, which conforms to its specifications.

The initialization signal to the USART (RESET) is provided programmatically by manipulation of bit 5 of port 2. It was necessary to place the reset of the 8251 under program control for two reasons. The first reason is that the MCS-48 does not supply a reset signal to other devices. The reason for this is that it was felt to be more useful to provide another pin of I/O function instead of a RESET OUT signal from the MCS-48. Although this situation could have been circumvented by the use of an externally generated reset which drove both the MCS-48 and the 8251, the second reason for program control of the reset to the USART still stands. The USART requires the presence of the CLK signal during reset in order to properly initialize itself. The ENT0 CLK instruction which the MCS-48 must execute before the 8251 will receive the CLK can obviously not be executed until after the system reset has ended. Reset of the USART can be accomplished by the following code segment:

```
        ENT0    CLK             ;TURN ON CLOCK
        ORL     P2, #00100000B  ;START RESET
        M0V     R2, #DELAY      ;DELAY USART
LOOP:   DJNZ    R2, LOOP        ;RESET TIME
        ANL     P2, #11011111B  ;END RESET
```

This code first enables the clock, then asserts the reset signal of a time period determined by the constant DELAY. The delay invoked is (10 + 5*DELAY) microseconds for DELAY >0. The USART requires a reset of approximately 6 CLK periods so DELAY is chosen to be 1 which ensures adequate reset timing. Note that for delays this short, NOP instructions could also be used to time the pulse.

The data clocks required by the USART are provided by the modem if the USART is operated in the synchronous mode. In the more common asynchronous mode, however, these clocks must be provided by circuitry associated with the 8251.

The 5.9904 MHz crystal was chosen because the resulting 1.9968 MHz clock to the USART can be evenly divided to provide transmit and receive clocks to the USART. Assuming the USART is in the x16 mode (i.e. it requires data clocks 16 times the baud rate) the 1.9968 MHz signal can be divided by 13 to generate the proper clock rate for 9600 baud operation. This 9600 baud clock can be further divided to give 4800, 2400, 1200, 600, and 300 baud signals. The 1200 baud signal can be divided by 11 to give a 109.1 baud signal which is within 1% of the 110 baud required by Teletypes.

The MCS-48 communicates with the 8251 in a memory mapped mode (i.e. as if the 8251 were external RAM). The instructions available to do this are MOVX ∂Rj, A which stores the contents of the accumulator at the external RAM location addressed by Rj (j=0 or 1), and its complement, the MOVX A, @ Rj instruction which moves data from the external RAM into the accumulator. Since the MCS-48 multiplexes addresses and data on the same eight bit bus an external latch would be required in order to address the USART with

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                    0 ; ---------------------------------
                    1 ; SERIAL TEST
                    2 ;  THIS CODE INTIALIZES THE USART
                    3 ;  AND TRANSMITS AN INCREMENTING
                    4 ;  PATTERN. HARDWARE SHOWN IF FIG 15.
                    5 ;---------------------------------
                    6
                    7 ; -------
                    8 ; EQUATES
                    9 ; -------
                   10
0020                11 MCLR    EQU     20H     ; USART RESET ADDRESS
0001                12 DLY     EQU     01H     ; USART RESET DELAY
007F                13 UCON    EQU     7FH     ; USART CONTROL ADDRESS
00CE                14 MODE    EQU     0CEH    ; USART MODE
0021                15 CMD     EQU     21H     ; USART CMD
007F                16 STAT    EQU     7FH     ; USART STATUS
0001                17 VAL     EQU     R1      ; TEST VALUE
00BF                18 MASK    EQU     0BFH    ; CHANGES CMD TO DATA CHANNEL
                    19
0100                20         ORG     100H
                    21                         ; TURN ON CLOCK
                    22                         ; AND RESET USART
0100 75             23 TEST:   ENTØ    CLK
0101 8A20           24         ORL     P2,#MCLR
0103 BA01           25         MOV     R2,#DLY
0105 EA05           26 LOOP:   DJNZ    R2,LOOP
0107 9ADF           27         ANL     P2,#(NOT MCLR)
                    28                         ; SELECT USART CONTROL
0109 237F           29         MOV     A,#UCON
010B 3A             30         OUTL    P2,A
                    31                         ; SEND MODE AND COMMAND
010C 23CE           32         MOV     A,#MODE
010E 90             33         MOVX    @R0,A   ; (CONTENTS OF R0 UNIMPORTANT)
010F 2321           34         MOV     A,#CMD
0111 90             35         MOVX    @R0,A
                    36                         ; DO FOREVER
                    37                         ;    SELECT USART STATUS
                    38                         ;       IF TXRDY=1 THEN
                    39                         ;       DO;
                    40                         ;          OUTPUT VALUE;
                    41                         ;          INCREMENT VALUE;
                    42                         ;       END;
                    43                         ;   END;
0112 237F           44 TLP:    MOV     A,#STAT
0114 3A             45         OUTL    P2,A
0115 80             46         MOVX    A,@R0   ; (CONTENTS OF R0 UNIMPORTANT)
0116 67             47         RRC     A
0117 E612           48         JNC     TLP
0119 F9             49         MOV     A,VAL
011A 9ABF           50         ANL     P2,#MASK
011C 90             51         MOVX    @R0,A
011D 19             52         INC     VAL
011E 2412           53         JMP     TLP
                    54                         ; END OF PROGRAM
                    55         END
```

Figure 16. 8251 Test Program

R0 or R1. In order to minimize the circuitry in Figure 15 an approach utilizing some of the I/O pins of the MCS-48 to address the 8251 was chosen instead. By connecting the chip select ($\overline{CS}$) input of the 8251 to bit 7 of port 2 (P27) and similarly connecting the C/$\overline{D}$ address line of the 8251 to bit 6 of port 2 (P26) it is possible to address the 8251 without using R0 or R1. The instruction sequence to access the 8251 is to first reset P27 and set P26 to the appropriate state, use a MOVX instruction to perform the appropriate operation, and then finally set P27 to deselect the 8251. As a concrete example of this addressing, Figure 16 shows the code necessary to initialize the 8251 and output an incrementing test pattern on a status driven basis.

If more than one 8251 were to be added to the MCS-48, or if other types of peripheral circuitry would be required (e.g. an 8253 timer to generate the data clocks) it would probably become desirable to add the circuitry necessary to use R0 or R1 to address the peripheral devices. The circuitry which has to be added to Figure 15 in order to make use of R0 or R1 to address the USART is shown in Figure 17. Note that only the changes to Figure 15 are shown. The additional component required is the 8212 eight bit latch. This latch is loaded, whenever a valid address is on the bus by the Address Latch Enable (ALE) signal provided by the MCS-48. During an external read or write cycle this address is used to address the 8251 in a linear select mode. In the circuit shown, the 8251 will be selected by any address with bit 1 a logical zero (XXXXXX0X) and the selection of control or data transfer (C/$\overline{D}$) will be based on bit zero of the address obtained from R0 or R1. Figure 18 shows the program of Figure 16 modified to utilize the addressing inherent in the MOVX instructions.

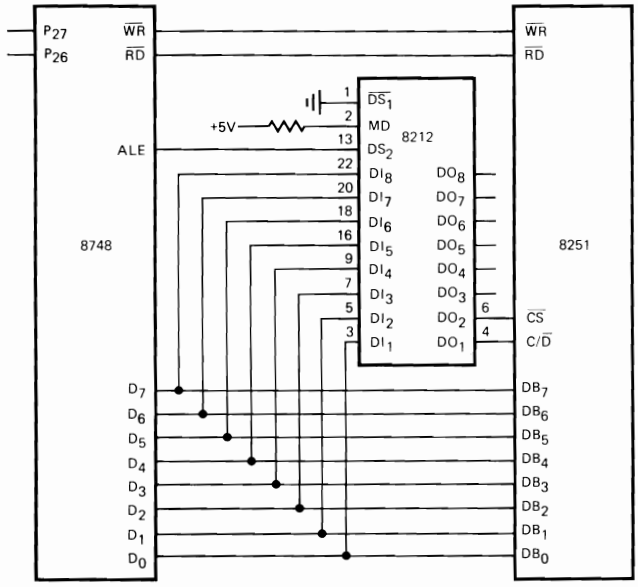


Figure 17. Modified MCS-48 to 8251 Interface

RECEIVING SERIAL CODE–A MORE SOPHISTICATED ALGORITHM

Although the USART does an admirable job of performing the serial I/O function for the MCS-48™, there are some situations where it can not be used. These situations may be caused by economic factors, such as an extremely cost sensitive design, or because the code which must be utilized cannot be accommodated by the USART. An example of of such a code will be discussed later. Recall that the principal objection to the approach to serial input shown in Figure 13 was that it consumes much of the processor's power by merely spinning in loops in order to wait preset time delays.

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                  0 ; ---------------------------------------
                  1 ; SERIAL TEST
                  2 ;  MIS CODE INTIALIZES THE USART
                  3 ;  AND TRANSMITS AN INCREMENTING
                  4 ;  PATTERN. HARDWARE SHOWN IF FIG 17.
                  5 ;---------------------------------------
                  6
                  7 ; -------
                  8 ; EQUATES
                  9 ; -------
                 10
0020             11 MCLR    EQU     20H      ; USART RESET ADDRESS
0001             12 DLY     EQU     01H      ; USART RESET DELAY
0003             13 UCON    EQU     03H      ; USARI CONTROL ADDRESS
00CE             14 MODE    EQU     0CEH     ; USARI MODE
0021             15 CMD     EQU     21H      ; USART CMD
0083             16 STAT    EQU     03H      ; USART STATUS
0001             17 VAL     EQU     R1       ; TEST VALUE
8080             18 DATA    EQU     80       ; USART DATA ADDRESS
                 19
0100             20         ORG     100H
                 21                          ; TURN ON CLLXK
                 22                          ; AND RESET USARI
0100 75          23 TEST:   ENT0    CLK
8101 8A20        24         ORL     P2,#MCLR
8103 BA21        25         MOV     R2,#DLY
0105 EA05        26 LOOP:   DJNZ    R2,LOOP
8107 9ADF        27         ANL     P2,#(NOT MCLR)
                 28                          ; SELECT USART CONTROL
8189 2303        29         MOV     A,#UCON
                 30                          ; SEND MODE AND COMMAND
0108 23CE        31         MOV     A,#MODE
010D 90          32         MOVX    @R0,A    ; (CONTENTS OF R0 UNIMPORTANT)
010E 2321        33         MOV     A,#CMD
0110 90          34         MOVX    @R0,A
                 35                          ; m FOREVER
                 36                          ;    SELECT USART STATUS
                 37                          :       IF TXRDY=1 THEN
                 38                          ;       DO;
                 39                                     OUTPUT VALUE;
                 40                                     INCREMENT VRUIE:
                 41                          ;       END;
                 42                          ;   END;
0111 2303        43 TLP:    MOV     A,#STAT
0113 80          44         MOVX    A,@R0    ; (CONTENTS OF R0 UNIMPORTANT)
0114 67          45         RRC     A
0115 E611        46         JNC     TLP
0117 F9          47         MOV     A,VAL
0118 B800        48         MOV     R0,#DATA
011A 90          49         MOVX    @R0,A
011B 19          50         INC     VAL
011C 2411        51         JMP     TLP
                 52                          ; END OF PROGRAM
                 53         END
```

Figure 18. Modified 8251 Test Program

The timer resident on the MCS-48 provides a solution to this problem. Instead of spinning in a loop the program can set the timer for a given interval, start it, and proceed to other tasks. When the timer overflows, an interrupt will be generated to notify the software that the present time period has elapsed. An extension of the algorithm of Figure 13 which uses the timer in this fashion in shown in Figure 19. This algorithm is identical to the preceding one up until the detection of the leading edge of the start bit. At this point the timer is set to one half of the bit time (P) and a return is made to the calling program which can start additional processing. At the completion of this time interval a timer overflow interrupt is generated. When the first interrupt is detected, the serial line is checked to ensure that it is in a spacing condition (valid START bit). If it is, the timer is set to P (to sample the middle of the first data bit) and a return is made to the program which was running when the interrupt occurred. If the serial line has returned to the MARK state, a status flag is set to indicate an error and a return is made. On subsequent interrupt detection, the data is sampled, the timer is reinitiated, and control is returned to the program which was running when the interrupt occurred. When the last (i.e. STOP) bit is detected a completion flag is set and a return is made to the program running when the timer overflow occurred. By periodically checking the error and completion flags the running program can determine when the interrupt driven receive program has a character assembled for it.

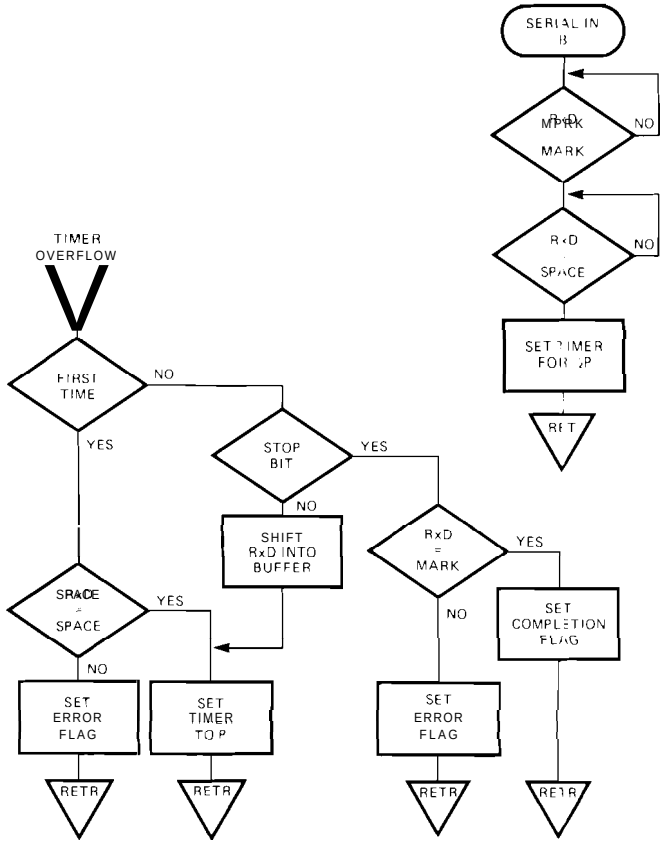


Figure 19. Improved Serial Input Routine

Using the timer to implement time delays as shown in Figure 19 results in considerable savings in processing time; two problems remain, however, which must be solved before an adequate software solution to the problem of receiving serial code can be found. The first problem is that even though the delays between bit samples are implemented via the timer rather than program loops the loop construction is still used to detect the leading edge of

the START bit. Although this results in the waste of processing power, the second problem is even more serious. For longer messages the required accuracy of the clocks becomes more and more stringent. Using the sampling technique discussed a cumulative error of one half a bit time in the time at which a bit sample is taken will result in erroneous reception. The maximum timing error which can be tolerated and yet still allow proper detection of an 11 bit ASCII character is then:

$$Emax = \frac{0.5*\text{BIT TIME}}{\text{CHARACTER TIME}} - \frac{0.5P}{11\,P} = 4.5\%$$

where P is the period of single bit. The corresponding calculation for a 32 bit character yields:

$$Emax = \frac{0.5P}{32\,P} = 1.6\%$$

Since Ihis calculation does not allow for distortion on the signals, it is obvious that either extremely stable clocks will be required or a more tolerant algorithm must be devised. This problem is particularly serious at relatively high baud rates where the resolution of the counter (80μsecs with a 6 MHz crystal) becomes a significant percentage of the period of the received signal. At the 110 baud rate of the Teletype the 80μsec resolution of the clock allows a maximum accuracy of 0.33%; at 2400 baud this figure is reduced to 3.8%.

Both efficient detection of the start bit and increased timing accuracy can be obtained if the MCS-48 can detect edges on the incoming received data (RxD). A hardware construct which allows this is shown in Figure 20.

The received data (RxD) is Exclusive NORed with bit seven of port two and fed into the TEST (T1) pin of the MCS-48. By manipulating P27 the program can now cause T1 to be either RxD or $\overline{RxD}$. (If P27 = 1 then T1 = RxD; if P27 = 0 then T1 = $\overline{RxD}$.) Note that not only can T1 be tested directly by the software but that it is the input which is used when the MCS-48 timer is in the event counter mode. The significance of this will be discussed later. The relationship between T1, P27, and RxD is given by the Boolean expression:

$$\overline{T1} = P27 \cdot \overline{RxD} + \overline{P27} \cdot RxD$$

Figure 21 flowcharts a means of utilizing this hardware construct to avoid the necessity of wasting time in program loops to detect the leading edge of the start bit. The receive operation is initialized when the program desiring to receive serial data calls the INIT subroutine (Figure 21a). Since INIT is going to manipulate the timer the first action it performs is to disable the timer overflow interrupt. Its next step is to set P27 to a logical 1. Setting P27 in this manner causes the TEST 1 input to the MCS-48 to follow $\overline{RxD}$. By setting up the receive circuitry in this manner a high to low transition will occur on TEST 1 when the RxD goes from the MARKING to SPACING state (i.e. the START

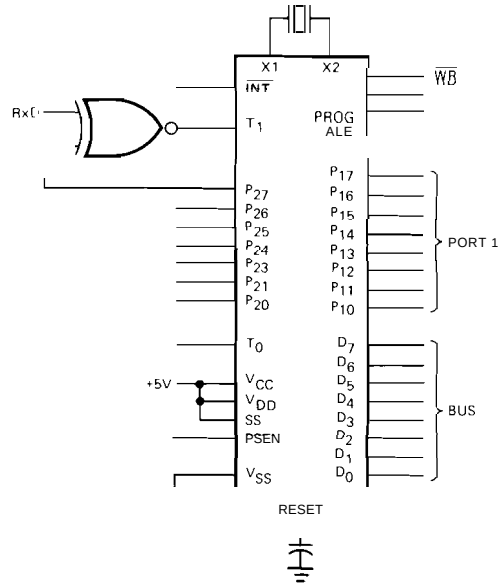


Figure 20. Detecting RxD Edges

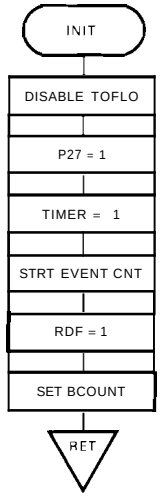


Figure 21a. Interrupt Driven Serial Receive Flowchart

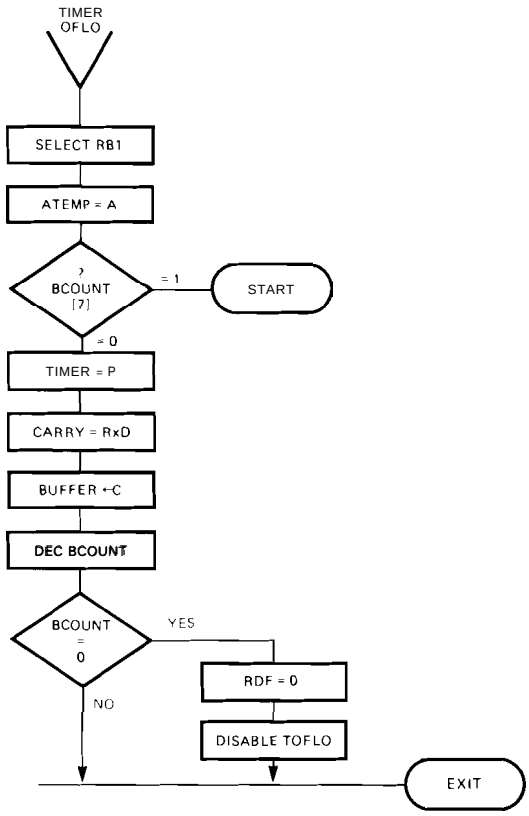


Figure 21b. Interrupt Driven Serial Receive Flowchart

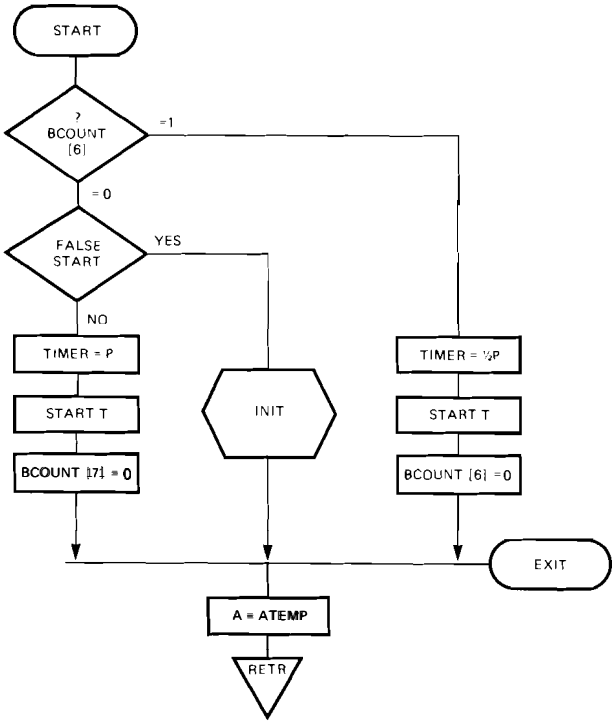
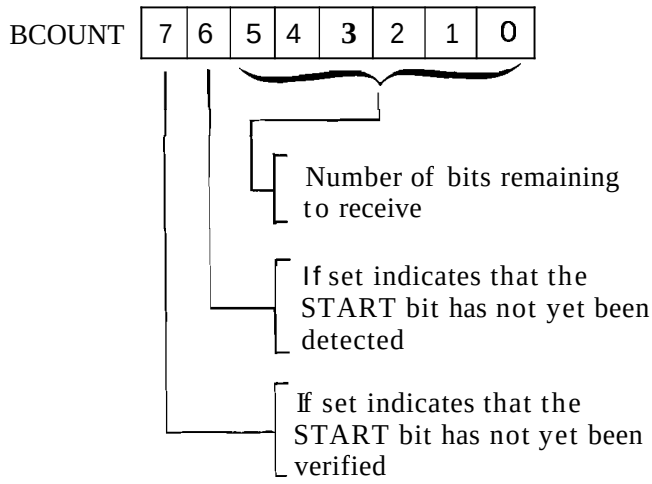


Figure 21c. Interrupt Driven Serial Receive Flowchart

bit occurs). By setting the timer to $0FF_H$ and enabling it in the event count mode, the INIT routine sets up the MCS-48 to generate a timer overflow interrupt on the next MARK to SPACE transition of RxD (the TEST 1 input doubles as the event counter input). Before returning to the calling program the INIT routine sets a flag (RDF) which will be cleared by the receive program when the requested receive operation is complete. INIT also sets a value into a register called BCOUNT. The receive program interprets BCOUNT as follows:



In order to request the reception of the 11 bit ASCII code INIT would set BCOUNT to 1100 1011B. The start bit has been neither verified nor detected and 11 bits (1011B) are required.

After INIT is called the reception of the individual serial data bits will proceed on an interrupt driven basis until a complete character has been assembled. When this occurs the interrupt driven program will set the RDF (Receive Done Flag) to a zero to indicate that it has completed the requested operation and then terminate itself. The procedure which is used to accomplish this is shown in Figures 21b and 21c.

Since all operations of this program are the result of the occurence of a timer overflow interrupt, it is necessary to briefly review the interrupt structure of the MCS-48. There are two sources of interrupt; an external interrupt which is the result of a logical zero signal applied to the INT pin of the MCS-48, and an internal interrupt which is caused by a timer overflow condition. The timer overflow occurs whenever the timer is incremented from $0FF_H$ to zero whether it be in the timer or event count mode. When one of these events occurs the hardware in the MCS-48 forces the execution of a CALL. This CALL has a preset address of location 3 if it is due to the external interrupt and location 7 if it is due to a timer overflow. If both of these

events occur simultaneously the external interrupt will take precedence. The CALL automatically saves the contents of the program counter for the running program and its PSW (program status word) on a stack the hardware maintains in RAM locations 8-23. Although the hardware saves the program counter and PSW, it remains the responsibility of any interrupt driven software to make absolutely certain that it does not modify any memory locations or registers which are being used by the main program. The most convenient way of ensuring this in the MCS-48 is to dedicate the second bank of registers (RBI) to the interrupt driven program. One of these registers has to be used to save the accumulator (which is not part of the register bank) but seven registers remain; including two which can be used as pointers to the rest of the RAM (RO and Rl). Note that if this approach is taken then these registers have to be allocated between the program which services the external interrupt and the one which services the timer overflow. This problem is somewhat alleviated by a hardware lockout which prevents the timer overflow interrupt from interrupting the external interrupt service routine and vice versa. This is implernented by locking out new interrupts between the tirne an interrupt is recognized and the time a RETR instruction is executed. The RETR instruction is like a normal RET (return from subroutine) except that the PSW as well as the program counter is restored. The RETR instruction can be very much thought of as a return from interrupt instruction in the MCS-48.

The receive program under discussion uses register bank 1 in the manner described. Whenever a timer overflow occurs (e.g. on the next MARK to SPACE transition of RxD after INIT is called), control is passed (by the hardware generated CALL) to the point labled TIMER OFLO in Figure 21b. This program segment immediately selects register bank 1 (RB1) and then saves the accumulator (A) in a location called ATEMP which is actually R7 of RB1. The program then tests bit seven of BCOUNT (R6 of RB1) to find out if a START bit has been verified (i.e. the edge of the START bit has first been detected and then verified to still be a SPACE one-half a bit time later. If BCOUNT [7] is a zero the START has been verified and the program proceeds to set the timer to P (the period of the serial bit), get the current serial data into the carry bit, and then shift the carry bit into a buffer. After saving the data the program decrements BCOUNT and tests it for zero. If BCOUNT is zero the receive operation is complete so the program sets RDF to a zero and disables timer overflow interrupts. Whether or not BCOUNT is zero, control is passed to EXIT where A is loaded with ATEMP and a RETR is executed. Note that since the state of the flip flop which selects RB1 is saved as part of the PSW, the execution of RETR automatically selects the register bank which was active when the interrupt occurred.

If BCOUNT [7] is still set when it is tested, control is passed to START (Figure 21c) where bit 6 is tested to determine if the START has been detected yet. If BCOUNT [6] is set it indicates that this is the first occurrence of a timer overflow since the receive process was initialized by the INIT subroutine. If this is so, the program assumes that the START bit has just started and therefore it sets the timer to one-half of a bit time (1/2 P), starts the timer in the timer mode, and clears BCOUNT [6] to indicate that the START bit has been detected. The next overflow will again result in the execution of the program in Figure 21b and again BCOUNT [7] will be found to be set. This time, however, BCOUNT [6] will be reset and the program will know that it should test the START bit to ensure that it is still a SPACE. This test is performed and if successful the timer is set for a bit period P and BCOUNT [7] is reset so that on the next occurrence of a timer overflow the program will know that it should start assembling serial bits into a character. If the test is unsuccessful, the subroutine INIT is used to reinitialize the receive program. In either case control is passed to EXIT where a return from interrupt mode occurs.

This receive program, listings of which appear in Figure 22, allows the reception of serial characters transparently to the main running software. After INIT is called the main program has only to check RDF periodically to find out if there is data in the buffer for it. It would be fairly easy to 'double buffer' this operation by providing a buffer which the receive program uses to deserialize the incoming code and a second buffer to store the assembled character. If the program would reinitialize itself upon completion, the reception of a string of characters could proceed in much the same way as it would if a status driven USART were being used.

Although this program solves the first problem of software controlled reception (lack of efficiency) the second problem—sensitivity to frequency variations—remains. An example of a code which would be susceptible to this problem is the 31,26 BCH code commonly used in supervisory control systems. (A supervisory control system is, in essence, a remote control system which allows a human or computer operator the control of a system via a serial communications link.) The BCH codes are used because of their error detection capabilities and are a class of cyclical redundancy

```
LOC  OBJ        SEQ     SOURCE STATEMENT

                  0
                  1 ; ***************************************
                  2 ;
                  3 ;       SERIAL INPUT USING M E MCS-48
                  4 ;        THIS CODE ASSUMES HARDWARE
                  5 ;        SHOWN IN FIG 20. TO USE
                  6 ;        THIS ROUTINE CALL INIT.
                  7 ;        WHEN RDF=0 THE ASSEMBLED
                  8 ;        CHARACTER WILL BE IN SERBUF
                  9 ;
                 10 ; ***************************************
                 11
                 12 ; --------
                 13 ; EQUATES
                 14 ; --------
                 15
0007             16 ATEMP   EQU     R7      ; STORAGE FOR A DURING INTERUPT
0006             17 BCOUNT  EQU     H6      ; CONTAINS NUMBER OF BITS IN MSG
0002             18 COUNT   EQU     R2      ; UTILITY COUNTER
0000             19 RX0     EQU     R0      ; POINTER
0008             20 BITNO   EQU     8       ; NUMBER OF BITS
0029             21 P       EQU     41      ; SAMPLE PERIOD
0020             22 SERBUF  EQU     20H     ; SERIAL BUFFER
0024             23 RDF     EQU     24H     ; RECEIVE DONE FLAG
                 24
                 25 ; ------------------------------------------
                 26 ; CONTROL PASSED HERE WEN TIMER OFW OCCURS
                 27 ; ------------------------------------------
                 28
                 29         ORG     07H
                 30                         : /*ENTER INTERRUPT MODE*/
0007 D5          31 IMVEC:  SEL     RB1
0008 AF          32         MOV     ATEMP,A
                 33                         ; IF BCOUNT[7]=0 THEN
0009 FE          34         MOV     A,BCOUNT
000A F223        35         387     START
                 36                         ; DO;
                 37                         ;    TIMER=P;
000C 23D7        38         MOV     A,#-P
000E 62          39         MOV     T,A
                 40                         ;    START TIMER
000F 55          41 SLLB:   STRT    T
                 42                         ;    /*CARRY=RXD*/
                 43                         ;    CARRY=P27 XNOR TEST1;
0010 0A          44         IN      A,P2

0011 F7          45         RLC     A
0012 5615        46         JT1     TISRD
0014 A7          47         CPJ     C
                 48                         ;    /*SHIFT CARRY INTO BUFFER*/
                 49                         ;    RX0=SERBUF;
                 50                         ;    RSHFT MEM(RX0);
0015 B820        51 TISRD:  MOV     RX0,#SERBUF
0017 20          52 SLOOP:  XLH     A,@RX0
0018 67          53         RRC     A
0019 20          54         XCH     A,@RX0
                 55                         ;    BCOUNT=BCOUNT-1;
                 56                         ;    IF BCOUNT=0 THEN
001A EE3F        57         WNZ     BCOUNT,SEXIT
                 58                         ;    DO;
                 59                         ;       RDF=0;
                 60                         ;       DISABLE EX IW;
                 61                         ;    END:
001C B824        62         MOV     RX0,#RDF
001E 27          63         CLR     A
001F A0          64         MOV     @RX0,A
0020 35          65         DIS     TCNTI
                 66                         : WD:
0021 043F        61         JMP     SEXIT
                 68                         ; ELSE
                 69                         ; DO;
                 70                         ;    IF BCOUNT[6]=0 THEN

0023 FE          71 START:  MOV     A,BCOUNT
0024 D237        72         JB6     SLLC
                 73                         ;    DO;
                 74                         ;       IF TEST1=0 THEN
0026 5635        75         JT1     SLLD
                 76                         ;       DO;
                 77                         ;          TIMER=P;
                 78                         ;          START TIMER;
                 79                         ;          P27=0;
                 80                                    EN I
                 81                                    BCOUNT[7]=0;
                 82                                 END;
0028 23D7        83         MOV     A, I-P
002A 62          84         MOV     T,A
002B 55          85         STRT    T
002C 9A7F        86         ANL     P2,#7FH
002E 05          87         En      I
002F FE          88         MOV     A,BCOUNT
0030 537F        89         ANL     A,#7FH
0032 AE          90         MOV     BCOUNT,A
0033 043F        91         JMP     SUIT
                 92                                 USE
                 93                                 DO;
                 94                                    CALL INIT;
                 95                                 END:
0035 1441        96 SLLD:   CALL INIT
                 97                         ;    ELSE
                 98                         ;    DO;
                 99                         ;       TIMER=P/2;
                100                         ;       START TIMER;
                101                         ;       BCOUNT[6]=0;
                102                         ;    END;
0037 23EC        103 SLLC:  MOV     A,#-(P/2)
0039 62          104        MOV     T,A
003A 55          105        STW     T
003B FE          106        MOV     A,BCOUNT
003C 53BF        107        ANL     A,#0BFH
003E AE          108        m v     BCOUNT,A
                 109                        ; END;
                 110                        ; /*EXIT INTERUPT MODE*/
003F FF          111 SEXIT: MOV     A,ATEMP
0040 93          112        RETR
                 113
                 114 ; ---------------------------
                 115 ; INITIALIZE ROUTINE-
                 116 ;       STARTS RECEIVE PROCESS
                 117 ; ---------------------------
                 118
                 119                        : INIT:
                 120                        ;    PROCEDURE;
                 121                        ;       m:
                 122                                 DISABLE INTERUPTS;
                 123                                 P27=1;
                 124                                 TIMER=-1;
                 125                                 START EVENT COUNT;
                 126                                 RDF=1;
                 127                                 BCOUNT=0C0H OR BITNO
                 128                              END;
                 129                        ;    END INIT;
0041 35          131 INIT:  DIS     TCNTI
0042 8A80        131        ORL     P2,#80H
0044 23FF        132        MOV     A,#-1
0046 62          133        MOV     T,A
0047 45          134        STRT    CNT
0048 B824        135        MOV     RX0,#RDF
004A B001        136        MOV     @RX0,#01H
004C B81E        137        MOV     RX0,#1EH       ; POINT AT BCOUNT
004E B0C8        138        MOV     @RX0,#(0C0H OR BITNO)
0050 25          139        EN      TCNTI
0051 83          140        RET
                 141                        ;END OF PROGRAM
                 142
                 143        END
```

Figure 22. Interrupt Driven Serial Receive Program

codes such as those used in synchronous data communications (e.g. BISYNC or SDLC). BCH codes, named for their originators Bose, Chaudhuri, and Hocquenghem, are characterized by having a length of $n=2^m-1$. The number of redundant check bits can be mt where t is a positive integer (clearly $mt \leqslant n$). The 31,26 code fits this format with m=5 and and t=1. The length of each message is $n=2^5-1=31$ with 5*1 redundant bits, leaving 26 bits available for data transmission. With an appropriate polynominal BCH codes can detect all errors consisting of 2t error bits and all burst errors of mt or fewer bits. The 31,26 BCH code will therefore detect any erroneous messages with 1 or 2 errors or bursts of errors of less than 5 bits. The 31,26 format (shown in Figure 23) requires the reception of a start bit followed by 31 information bits, clearly beyond the capability of the USART but perhaps within reach of a program controlled approach using the MCS-48 itself.

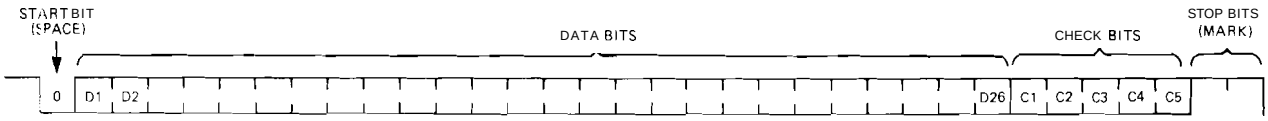


Figure 23. 31,26 BCH Code

A concept which reduces sensitivity to frequency deviations and thus allows the reception of longer codes is shown pictorially in Figure 24. The first line of this timing chart shows an alternative ones and zeros pattern on the RxD with a period of 5 milliseconds. The second line shows that by sampling at a period of exactly 5 milliseconds the data can be properly interpreted. The third and fourth lines show the effects of sampling with a period of six and four milliseconds respectively. In either case, an error occurs at the third sample where both periods result in sampling on an edge of the RxD signal. The third line of Figure 24 shows a hybrid sampling scheme which, based on some additional information, switches sampling periods between the two values. As can be seen in Figure 24, the data is sampled with a 4 millisecond period until the sampling begins to fall behind the data; st this point the sampling period is increased to six milliseconds and the sampling first catches up and then passes the center point of the data. As soon as this happens, the sampling period reverts to the 4 millisecond period and the cycle repeats. It can be seen that this scheme sets up a pattern which repeats indefinitely and the data can be successfully sampled. Note that the sampling pattern established is alternating periods of four and six milliseconds. The average period of this pattern, as might be expected, is 5msec. Line 5 of Figure 24 shows the effect of a change in transmission speed to a period of 5.5 msec with no change in the sampling time. The sampling is again successful but the new sampling pattern is 4-6-6-6; 4-6-6-6, etc. Note that the average sample is again equal to the period of the received data (5.5). While this scheme does seem to work, the question of what additional information is needed remains.

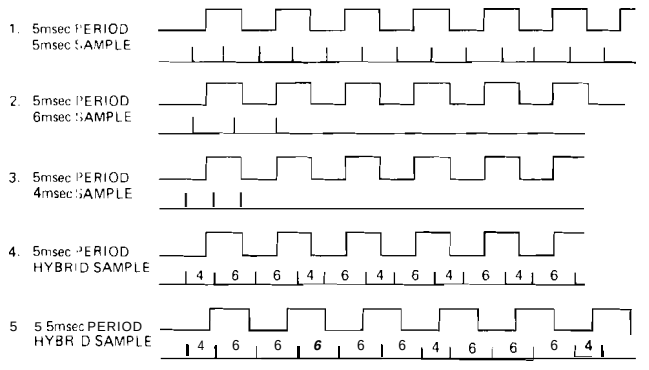


Figure 24. Various Sampling Alternatives

The MSC-48 must somehow decide when it is drifting out of synchronization and take corrective action. By referring back to Figure 24 it can be seen that if the MCS-48 could determine where the edges of RxD occurred with respect to its sampling times then the additional information would be available. As can be seen in the figure the choice of sampling period can be based on the following rule:

> *If an edge on the RxD line occurs during the first half of the current sampling period, then use the short period for the next sample. If an edge occurs during the second half of the period, then use the long sampling period for the next sample.*

If the data on the RxD line does not change, of course, the MCS-48 will drift out of synchronization just as the original algorithum did. As long as edges occur on TxD, however, synchronization can be maintained. To maximize the allowable time between edges, the following addition could be made to the above rule:

> *If no edge occurs on the RxD line during a sample, then change sanzpling period from short to long or vice versa.*

Note that this addition to the rule will result in using an average of the two sampling periods when no edge occurs for several bit times.

The edges of RxD can be easily detected by the use of the same structure (the Exclusive – NOR gate) which was added to the MCS-48 in Figure 20. This gate, which is used to detect the edge on RxD which begins the START bit, can naturally be used to detect any edge. Since the timer is being used to time the bit period, however, the event count input (T1) is not useful during the receive itself. By connecting the output of this gate, however, to the $\overline{\text{INT}}$ input to the MCS-48 (see Figure 25) it is possible to detect edges on RxD with the event counter when the program is trying to detect the START bit and by the external interrupt when the program is using the timer to control the sampling times.

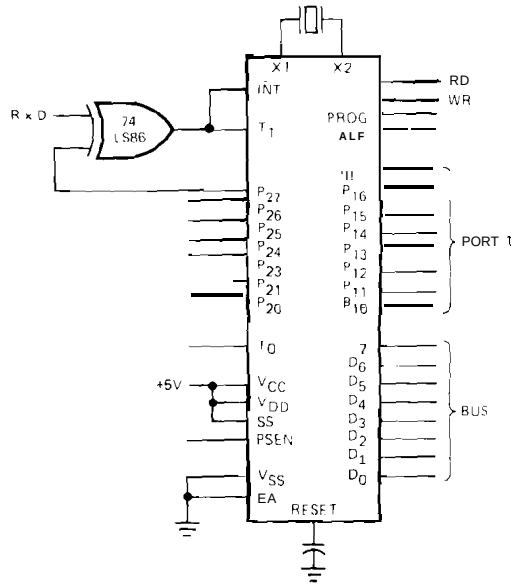
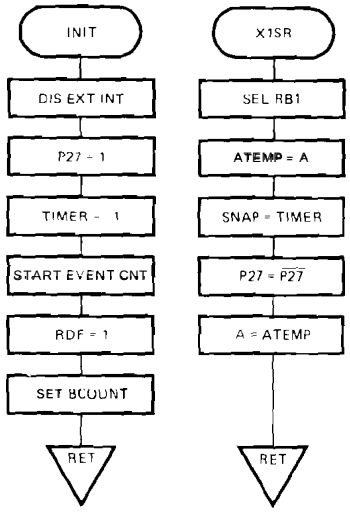


Figure 25. Modified Edge Detection

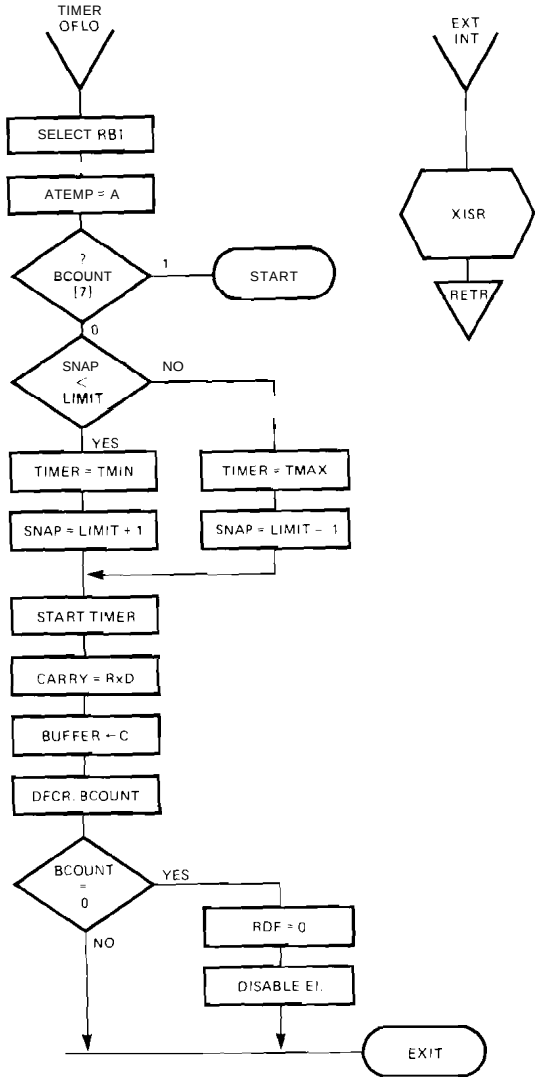
A modification to the program of Figure 21 which implements this new sampling algorithm is shown in Figure 26. The first deviation from the original program is the addition of a routine (XISR, Figure 26a which is called when an external interrupt occurs (i.e. when an edge occurs on RxD). This routine saves the status of the running program and then stores the current value of the timer register in a location called SNAP (R5 of RBI). After doing these operations the program complements bit 7 of port 2. Manipulating P27 in this manner will cause the Exclusive NOR gate to turn off the external interrupt and will set it up to generate another interrupt when the RxD line changes again (has another edge).



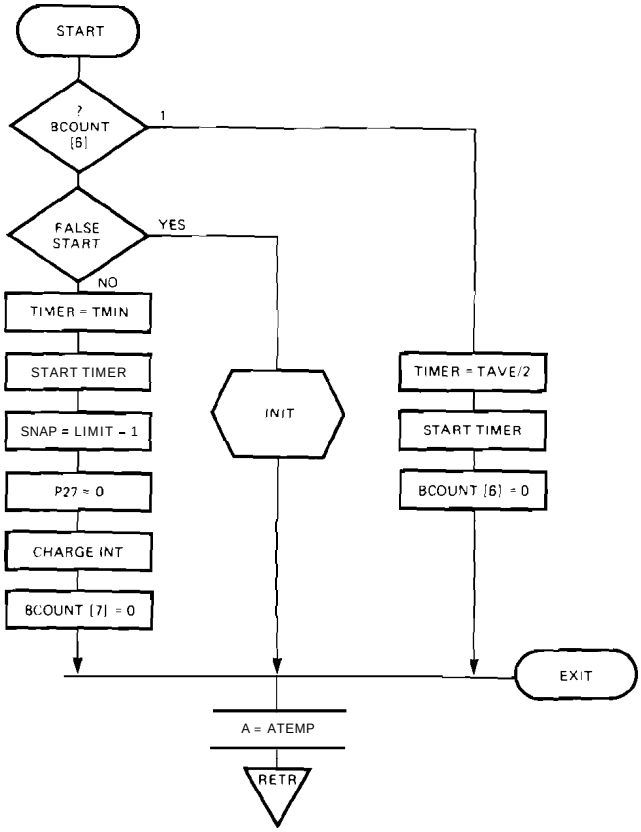
Hybrid Sampling Flowchart

Because of this edge detection it is important to condition RxD with hardware filters to ensure that the edges of RxD are clean. Any ringing will cause repeated CALLs to XISR and probable erroneous operation. The changes to the START process (Figure 26c) are two-fold; first the TIMER is set to one half the average of the two sample periods when the START bit is first detected (BCOUNT [6] = 1), and second the processing of the edge information is initialized by presetting SNAP and clearing P27.

SNAP is preset so that when the reception of data actually begins (Figure 26b BCOUNT [7] = 0), the decision block which tests SNAP against LIMIT will be initialized. This block actually compares the value in SNAP with a LIMIT value which is used to determine if the sampling point is ahead or behind the actual midpoint of the serial data. If the sampling is ahead then the timer is set for TMIN; if the sampling is behind then the timer is set for



Hybrid Sampling Flowchart



Hybrid Sampling Flowchart

TMAX. By presetting SNAP in the manner shown in the flowcharts the second rule of the algorithm, (if *no edge appears on the RxD line during a sample, then change the sampling periods short to long or vice versa)* is automatically met. If an edge occurs then XISR will modify SNAP, if XISR is not invoked between two samples then the choice of timer periods will alternate. The only other significant change to the algorithm is that the INIT routine must now lock out all interrupts, not just the timer overflow interrupt, while it is operating. A program which uses this algorithm to receive a 32 bit message is shown in Figure 27.

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                    0
                    1 ; ************************************
                    2 ;
                    3 ;       SERIAL INPUT USING MCS-48
                    4 ;         THIS CODE ASSUMES HARDWARE
                    5 ;         SHOWN IN FIG 25.  PROGRAM
                    6 ;         IS SIMULAR TO PREVIOUS
                    7 ;         ONE, A MORE SOPHISTICATED
                    8 ;         SAMPLING ALGORITHM IS USED
                    9 ;
                   10 ; NOTE:  A PL/M LIKE LANGUAGE WAS USED
                   11 ;        TO COMMENT THIS LISTING AND
                   12 ;        SEVERAL OTHERS IN THIS NOTE, NO
                   13 ;        COMPILER EXISTS FOR THE MCS-48.
                   14 ;        THE COMMENTS WERE 'HAND
                   15 ;        COMPILED' INTO ASSEMBLY CODE
                   16 ;;
                   17 ; ************************************
                   18
                   19 ; --------
                   21 ; EQUATES
                   21 ; --------
                   22
0007               23 ATEMP   EQU     R7      ; STORAGE FOR A DURING INTERUPT
0006               24 BCOUNT  EQU     R6      ; CONTAINS NUMBER OF BITS IN MSG
0005               25 SNAP    EQU     R5      ; TAKES TIMER SNAP SHOT ON RXD EDGE
0002               26 COUNT   EQU     R2      ; UTILITY COUNTER
0000               27 RX0     EQU             ; POINTER
0020               28 BITNO   EQU     32      ; NUMBER OF BITS
0014               29 LIMIT   EQU     21      ; TEST VALUE FOR MIN/MAX SAMPLING
FFD5               30 TMAX    EQU     -43     ; MAX SAMPLE PERIOD
FFD9               31 TMIN    EQU     -39     ; MINIMUM SAMPLE PERIOD
FFEC               32 HALF    EQU     -20     ; HALF NOMINAL PERIOD
0020               33 SERBUF  EQU     20H     ; START OF SERIAL BUFFER
0024               34 RDF     EQU     24H     ; RECEIVE DONE FLAG
                   35
                   36 ; ---------------------------
                   37 ; CONTROL PASSED HERE ON EXT. INT.
                   38 ; ---------------------------
                   39
0003               40         ORG     03H
                   41                         ; CALL SERVICE ROUTINE
0003 1466          42 EIVEC:  CALL    XISR
0005 93            43         RETR
                   44
                   45 ; ------------------------------------
                   46 ; CONTROL PASSED HERE WHEN TIMER OFLO OCCURS
                   47 ; ------------------------------------
                   48
                   43                         ; /*ENTER INTERUPT MODE*/
0006 D5            50 TMVEC:  SEL RB1
0007 AF            51         MOV     ATEMP,A
                   52                         ; IF BCOUNT[7]=0 THEN
0008 FE            53         MOV     A,BCOUNT
0009 F236          54         JB7     START
                   55                         ; DO;
                   56                         ;    IF SNAP<LIMIT THEN
000B FD            57         MOV     A,SNAP
000C 6314          58         ADD     A,#LIMIT
000E F217          59         JB7     SLLA
                   61                         ;    DO;
                   61                         ;       TIMER=TMIN;
                   62                         ;       SNAP=LIMIT+1;
                   63                         ;    END;
0010 23D9          64         MOV     A,#TMIN
0012 62            65         MOV     T,A
0013 BD13          66         MOV     SNAP,#LIMIT-1
0015 641C          67         JMP     SLLB
                   68                         ;    ELSE
                   69                         ;    DO;
                   70                         ;       TIMER=TMAX;
                   71                         ;       SNAP=LIMIT-1;
                   72                         ;    END;
0017 23D5          73 SLLA:   MOV     A,#TMAX
```

Figure 27. Hybrid Sampling Program

```
LOC  OBJ       SEQ        SOURCE STATEMENT

0019 62          74         MOV   T,A
001A BD13        75         MOV   SNAP,#LIMIT-1
                 76                   ;    START TIMER;
001C 55          77 SLLB:   STRT  1
                 78                   ;    /*CARRY=RXD*/
                 79                   ;    CARRY=P27 XOR TEST1;
001D DA          80         IN    A,P2
001E F7          81         RLC   A
001F 4622        82         JNT1  TISRL
0021 A7          83         CPL   C
                 84                   ;    /*SHIFT CARRY INTO BUFFER*/
                 85                   ;    RX0=SERBUF;
                 86                   ;    COUNT=4;
                 87                   ;    DO WHILE COUNT<>0;
                 88                   ;       RSHFT MEM(RX0);
                 89                   ;       RX0=RX0+1;
                 90                   ;       COUNT=COUNT-1;
                 91                   ;    END;
0022 B820        92 TISRL:  MOV   RX0,#SERBUF
0024 BA04        93         MOV   COUNT,#4
0026 28          94 SLOOP:  XCH   A,RX0
0027 67          95         RRC   A
0028 20          96         XCH   A,@RX0
0029 18          97         INC   RX0
002A EA26        98         WNZ   COUNT,SLOOP
                 99                   ;    BCOUNT=BCOUNT-1;
                100                   ;    IF BCOUNT=0 THEN
002C EE54       101         DJNZ  BCOUNT,SEXIT
                102                   ;    W:
                103                   ;       RDF=0;
                104                   ;       DISABLE EX INT;
                105                   ;    END;
002E B824       106         MOV   RX0,#RDF
0030 27         107         CLR   A
0031 A0         108         MOV   @RX0,A
0032 35         109         DIS   TCNTI
0033 15         110         DIS   I
                111                   ; END;
0034 0454       112         JMP   SEXIT
                113                   ; USE
                114                   ; DO;
                115                   ;    IF BCOUNT(6)=0 THEN
0036 FE         116 START: MOV   A,BCOUNT
0037 D24C       117         JB6   SLLC
                118                   ;    DO;
                119                   ;       IF TEST1=0 THEN
0039 564A       120         JT1   SLLD
                121                   ;       DO;
                122                   ;          TIMER=TMIN;
                123                   ;          START TIMER;
                124                   ;          SNAP=LIMIT+1;
                125                   ;          P27=0;
                126                   ;          EN I
                127                   ;          BCOUNT(7)=0;
                128                   ;       END;
003B 23D9       129         MOV   A,#TMIN
003D 62         130         MOV   T,A
003E 55         131         STRT  T
003F BD15       132         MOV   SNAP,#LIMIT+1
0041 9A7F       133         ANL   P2,#7FH
0043 05         134         EN    I
0044 FE         135         MOV   A,BCOUNT
0045 537F       136         ANL   A,#7FH
0047 AE         137         MOV   BCOUNT,A
0048 0454       138         JMP   SEXIT
                139                   ;       ELSE
                140                   ;       DO;
                141                      CALL INIT;
                142                   ;       END;

LOC  OBJ       SEQ        SOURCE STATEMENT

004A 1456       143 SLID:  CALL INIT
                144                   ;    USE
                145                   ;    W:
                146                   ;       TIMER=(TMIN+TMAX)/2;
                147                   ;       START TIMER;
                148                   ;       BCOUNT(6)=0;
                149                   ;    END;
004C 23EC       150 SLLC:  MOV   A,#HALF
004E 62         151        MOV   T,A
004F 55         152        STRT  T
0050 FE         153        MOV   A,BCOUNT
0051 53BF       154        ANL   A,#0BFH
0053 AE         155        MOV   BCOUNT,A
                156                   ; END;
                157                   ; /*EXIT INTERUPT MODE*/
0054 FF         158 SUIT;  MOV   A,ATEMP
0055 93         159        RETR
                160
                161 ; ----------
                162 ; INTIALIZE ROUTINE-
                163 ;       STARTS RECEIVE PROCESS
                164 ; ----------
                165
                166                   ; INIT:
                167                   ;    PROCEDURE;
                168                   ;       DO;
                169                          DISABLE INTERUPTS;
                170                          P27=1;
                171                          TIER=-1;
                172                          START EVENT COUNT;
                173                          RDF=1;
                174                          BCOUNT=0C0H OR BITNO
                175                       END;
                176                   ;    END INIT;
0056 15         177 INIT:  DIS   I
0057 35         178        DIS   TCNTI
0058 8A80       179        ORL   P2,#80H
005A 23FF       180        MOV   A,#-1
005C 62         181        MOV   T,A
005D 45         182        STRT  CNT
005E B824       183        MOV   RX0,#RDF
0060 F9         184        MOV   A,@1
0061 A0         185        MOV   @RX0,A
0062 25         186        EN    TCNTI
0063 BEE0       187        MOV   BCOUNT,#0C0H OR BITNO
0065 83         188        RET
                189
                190
                191 * ----------
                192 ; INTERUPT SERVICE ROUTINE
                193 ; ----------
                194                   ; XISR:
                195                   ;    PROCEDURE;
                196                   ;    DO;
                197                   ;       /*ENTER INTERRUPT MODE*/
                198                   ;       SNAP=TIMER;
                199                   ;       P27=NOT P27;
                200                   ;    END XISR;
0066 D5         201 XISR:  SEL   RB1
0067 AF         202        MOV   ATEMP,A
0068 42         203        MOV   A,T
0069 AD         204        MOV   SNAP,A
006A 0A         205        IN    A,P2
006B D380       206        XRL   A,#80H
006D 3A         207        OUTL  P2,A
006E FF         208        MOV   A,ATEMP
006F 83         209        RET
                210                   ; END OF PROGRAM
                211        END
```

Figure 27. Hybrid Sampling Program

TRANSMITTING SERIAL CODE

Serial transmission is conceptually far simpler than serial reception since no synchronization is required. All that is required is to use the timer to generate interrupts at the bit rate and present the character to be transmitted serially at an I/O pin. A program which does this is shown in Figure 28. The transmission of serial data becomes much more complicated if it must occur simultaneously with reception.

If both reception and transmission are to occur simultaneously then obviously contention will exist for the use of the timer. It is possible to allow the simultaneous reception and transmission of serial data using the timer as a general clock which conti-01s software maintained timers. The attainable baud rates using such techniques are, however, limited and the use of a 8251 USART is probably indicated in all but the most cost sensitive applications. An exception to this rule occurs when the system, although full duplex in nature, actually transmits the same data as it receives. An example of this is a microprocessor driving a terminal such as a Teletype. Although the circuit to the terminal is full duplex, the data that is transmitted is generally the same as that received. A minor modification to the program shown in Figure 26 would implement this mode of operation. The modification would be to the XISR routine and it would add the code necessary to place the TxD I/O pin in the same state as the RxD line. Since any change in RxD results in a call to XISR, this modification would cause the retransmission of any received data. Whenever it becomes necessary to transmit data which is not being received, the program of Figure 28 could be used in a half duplex manner.

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                  1 ;-----------------------------------
                  2 ; SERIAL TRANSMIT ON THE MCS48
                  3 ;   TO USE PUT A CHAR IN BUFF AND
                  4 ;   SET CHARAV TO 0FFH. WHEN THE
                  5 ;   TRANSMITTER IS READY FOR ANOTHER
                  6 ;   CHAR IT WILL CLEAR CHARAV. THE
                  7 ;   TRANSMISSION IS DOUBLE BUFFERED.
                  8 ;-----------------------------------
                  9
                 10 ; -------
                 11 ; EQUATES
                 12 ; -------
                 13
0007             14 ATEMP   EQU     R7       ; STORAGE FOR A DURING INT.
0006             15 PTOS    EQU     R6       ; PARALLEL TO SERIAL CONVERTER
0005             16 BUFF    EQU     R5       ; CHARACTER BUFFER
0004             17 CHARAV  EQU     R4       ; CHARACTER AVAILABLE FLAG
0003             18 COUNT   EQU     R3       ; BIT COUNTER
00EF             19 CBIT    EQU     0EFH     ; MASK TO CLEAR TXD IN P24
0010             20 SBIT    EQU     010H     ; MASK TO SET TXD IN k24
FFE7             21 P       EQU     -41      ; PERIOD OF TXD
                 22
                 23 ;-----------------------------------
                 24 ; CONTROL PASSED HERE ON TIMER OVERFLOW
                 25 ;-----------------------------------
0007             26         OK:     07H
                 27                         ; ENTER INTERUPT MODE
0007 D5          28 TOFLO:  SEL     RB1
0008 AF          29         MOV     ATEMP,A
                 30                         ; SET TIMER FOR P
0009 2307        31         MOV     A,#P
000B 62          32         MOV     T,A
000C 55          33         STRT    T
                 34                         ; GET BIT INTO CARRY
000D 141D        35         CALL    BIT
                 36                         ; SEX TXD 10 CARRY
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

000F 0A          37         IN      A,P2
0010 D380        38         XRL     A,#80H
0012 3A          39         OUTL    P2,A
0013 F619        40         JC      BITON
0015 9AEF        41         ANL     P2,#CBIT
0017 041B        42         JMP     EXIT
0019 8A10        43 BITON:  ORL     P2,#SBIT
001B FF          44 EXIT:   MOV     A,ATEMP
001C 93          45         RETR
                 46
                 47 ;-----------------------------------
                 48 ; BIT ROUTINE
                 49 ; -PICKS THE NEXT BIT TO TRANSMIT
                 50 ;-----------------------------------
                 51
001D FB          52 BIT:    MOV     A,COUNT
001E C627        53         JZ      IDLE
0020 FE          54         MOV     A,PTOS
0021 67          55         RRC     A
0022 4380        56         ORL     A,#80H
0024 AE          57         MOV     PTOS,A
0025 CB          58         DEC     COUNT
0026 83          59         RET
                 60
0027 97          61 IDLE:   CLR     C
0028 FC          62         MOV     A,CHARAV
0029 962B        63         JNZ     GOTONE
002B A7          64         CPL     C
002C 83          65         RET
                 66
002D FD          67 GOTONE: MOV     A,BUFF
002E AE          68         MOV     PTOS,A
002F BB0A        69         MOV     COUNT,#10
0031 BC00        70         MOV     CHARAV,#0
0033 83          71         RET
                 72                         ; END OF PROGRAM
                 73         END
```

Figure 28. Serial Transmission

GENERATING PARITY

Many communications schemes require the generation and checking of parity. If a USART is used it can be programmed to automatically generate and check parity. If the communications is handled by software within the MCS-48™ then the program must perform parity calculations. Calculating parity is easy if one remembers what parity really means. A character has even parity if the number of one bits in it is even. A character has odd parity if it has an odd number of ones. The program segment shown in Figure 29 can be caused to calculate parity. It starts by setting a loop count to eight and

```
LOC  OBJ        SEQ         SOURCE STATEMENT

                  0
                  1
                  2 ; **************
                  3 ;
                  4 ; PARITY
                  5 ;   THIS PROGRAM GENERATES PARITY
                  6 ;   ON THE ACCUMULATOR
                  7 ;     CARRY WILL BE SET IF A HAS ODD PARITY
                  8 ;
                  9 ; **************
                 10
                 11
                 12 ; --------
                 13 ; EQUATES
                 14 ; --------
                 15
0002             16 COUNT   EQU     R2
                 17
0100             18 PAR:    ORG     100H
0100 BA08        19         MOV     COUNT,#8         ; SET LOOP COUNT
0102 97          20         CLR     C                ; INITIALIZE CARRY
                 21                                  ; FOR EACH ZERO BIT IN A
                 22                                  ; COMPLEMENT THE CARRY FWG
0103 77          23 LOOP:   RR      A
0104 1207        24         JB0     OVER
0106 A7          25         CPL     C
0107 EA03        26 OVER:   DJNZ    COUNT,LOOP
                 27                                  ; END OF PROGRAM
                 28         END
```

Figure 29. Parity Generation

clearing the CARRY flag. After this initialization a loop is executed eight times. During each execution the accumulator is rotated and the least significant bit is tested. If the bit is a zero the CARRY flag is complemented, if the bit is a one no further action is taken. Since an even number of zeros implies an even number of ones for an eight bit character, after all eight loops have been accomplished the CARRY bit will be set if an odd number of ones were encountered; it will be reset if the number were even. Since the RR instruction does not involve CARRY the net result of executing this program loop is to set CARRY if parity is odd without effecting the character in the accumulator.

CONCLUSION

This Application Note has presented a very small sampling of the application techniques possible with the MCS-48™ family. The application of this new single chip computer system to tasks which have not yet yielded to the power of the microprocessor will present a fascinating challenge to the system designer.

3065 Bowers Avenue
Santa Clara California 95051
Tel 14081 246-7501
TWX 910-338-0026
TELEX 34-6372

MICROCOMPUTER AND MEMORY COMPONENT SALES AND MARKETING OFFICES

AUGUST 1977

U.S. AND CANADIAN SALES OFFICES

ALABAMA
Glen While Associates
7844 Horseshoe Trail
Huntsville 35802
Tel (205) 883 9394

ARIZONA
Sales Engineering Inc
7226 Stetson Drive, Suite
Scottsdale 85252
Ter (602) 945-5781
TWX 910-950-1288

Intel Corp.
8650 N 35th Avenue
Phoenix 85021
Tel: (602) 242-7205

CALIFORNIA
Inte Corp *
990 E. Arques Ave.
Suite 112
Sunnyvale 94086
Tel: (408) 738-3870
TWX. 910-339-9279
TWX 910-338-0255

Mac I
P O Box 1420
Cupertino 95014
Tel (408) 257 9880

Earle Associates Inc
4805 Mercury Street
Suite L
San Diego 92111
Tel (714) 278-5441
TWX 910-335-1585

Mac-I
P O Box 8763
Fountain Valley 92708
Tel (714) 839-3341

Intel Corp:
1651 *East 4th* Street
Suite 228
Santa Ana 02701
Tel (714) 835-9842
TWX 910-595-1114

COLORADO
Intel Corp.
12075 East 45th Avenue
Suite 31C
Denver 80239
Tel (303) 373-4920
TWX: 910-932-0322

CONNECTICUT
Intel Corp.
Peacock Alley
1 Padanaram Road
Danbury 06810
Tel: (203) 792 8366

FLORIDA
Intel Corp
2020 W. McNab Road. Suite 104
Ft. Lauderdale 33309
Tel (305) 971-7200
TWX 510-956-9407

Intel Corp.
5151 Adanson Street. Suite 105
Orlando 32804
Tel: 1305) 628-2393
TWX 810-853 9219

ILLINOIS
Intel Carp *
1000 Jorie Boulevard
Suite 224
Oakbrook 60521
Tel (312) 325-9510
TWX 910-651-5881

IOWA
Technica Representatives, Inc
1703 Hillside Drive N, W
Cedar Rapids 52405
Tel: (319) 396-5662

KANSAS
Technical Representatives Inc
801 Clairborne
Olathe 66061
Tel (913) 782-1177
TWX. 910-749-6412

MARYLAND
Glen White Associates
57 West Timonium Road
Timonium 21093
Tel (301) 252-6360

Intel Corp *
57 West Timonium Road
Suite 307
Timonium 21093
Tel: (301) 252-7742
TNX 7'0-232 1807

MASSACHUSETTS
Intel Corp'
187 Billerica Road Suite 14A
Chelmsford 01824
Tel 1617) 256-6567
TWX 710-343-6333

MICHIGAN
Intel Corp.
26500 Northwestern Hwy
Suite 401
Southfield 48075
Tel: (313) 353-0920
THX 910-420-1212
TELEX 2.1143

MINNESOTA
Intel Corp
8200 Normandae Avenue
Suite 422
Bloomington 55637
Tel (612) 835-6722
TWX 910 576-2867

MISSOURI
Technical Representatives, nc
Trade Center Bldg
300 Brookes Drive Suile 108
Hazelwood 63042
Tel: (314) 731-5200
TWX 913-762-0618

NEW JERSEY
Intel Corp
7 Kilmer Road
Edison 08817
Tel: (201) 985-9100
TWX 710-480-6238

NEW YORK
Irtel Corp *
350 Vanderbilt Motor Pkwy.
Suite 402
Hauppauge 11787
Tel: (516) 231-3300
TWX 510-221-2198

Intel Corp
474 *Thurston* Road
Rochester, N.Y. 146;9
Tel (716) 398-7340
TWX 510-253-3841

NEW YORK (cont.)
T-Squared
4054 Newcourt Ave
Syracuse 13206
Tel: (315) 463-8592
TWX 710 541-0554

T-Squared
640 Kreag Rd
P.3. Box W
Pittsford 14534
Tel: (716) 381-2551
TELEX. 97-8289

Intel Corp.
85 Market Street
Poughkeepsie, New York 12601
Tel: (914) 473-2303
TWX. 510-248-0060

NORTH CAROLINA
Glen While Associates
3700 Computer *Dr.* Suite 330
Raleigh 27609
Tel (919) 787-7016

OHIO
Intel Corp *
8312 North Main Street
Dayton 45415
Tel (513) 890-5350
TELEX *288-064*

Intel Corp'
26250 *Euclid* Ave
Suite 531F
Euclid 44132
Tel: (216) 289-0101

PENNSYLVANIA
Intel Corp.*
520 Pennsylvania Ave
Fort Washington 19034
Tel (215) 542.9444
TWX 510-661-0709

TENNESSEE
Glen White Associates
Rt. #12 Noorwood S/D
Jonesboro 37659
Tel 16151926-0184

Glen White Associates
2523 Howard Road
Germantown 38138
Tel (901) 754-0483

TEXAS
Mycrosystems Marketing Inc
13777 N Central Expressway
Suite 405
Dallas 75231
Tel (214) 238-7157
TWX: 910-867-4763

Mycrosystems Marketing Inc
6610 Harwin Avenue, Suite '25
Houston 77038
Tel (713) 783-2900

Mycrosystems Marketing In:
2622 Geronimo Trail
Austin 78746
Tel. 15121 266-1750

Inlel Corp *
2925 L.B J Freeway
Dallas 06234

Tel: (2141 241-9521
TWX 910-860-5487

VIRGINIA
Glen White Associates
P O Ecx 1104
Lynchburg 24505
Tel (8041 384-6920

WASHINGTON
E S Chase Co
P O Box 80933
Seattle 98108
Tel (206) 762-4824
Twx 910 444 2298

CANADA
Intel Corp.
70 Chamberlain Ave
Ollawa, Ontario K1S 1V9
Tel (613) 232-8576
TELEX 053-4419

Multitek Inc *
4 Barran Street
Ottawa, Ontario K2J 1G2
Tel 16131 825-4553
TELEX 053 1585

EUROPEAN MARKETING OFFICES

BELGIUM
Intel International*
Rue du Moulin a Papier
51-Boite 1
B 1160 Brussels
Tel (02) 660 30 10
TELEX 24814

FRANCE
Intel Corporation S A R L *
74. Rue D'Arcueil
Silic 223
34528 *Rungis Cedex*
Tel: (0') 687 22 21
TELEX 270475

SCANOINAVIA
Intel Scandinavia A/S*
Lyngbyvej 32 2nd Floor
DK-2100 Copenhagen East
Denmark
Tel: (01) 18 20 00
TELEX. 19567

Intel Sweden AB*
Box 20092
S-16120 Bromma
Sweden
Tel: (08) 98 53 90
TELEX 12261

ENGLAND
Intel Corporation (U.K.) Ltd.*
Broadfield House
4 Between Towns Road
Cowley, *Oxford* OX4 3NB
Tel: (0865) 77 14 31
TELEX 837203

Intel Corporation (U K) Ltd
46-50 Beam Street
Nantwich, Cheshire CW5 5LJ
Tel 102701 62 65 60
TELEX 36620

GERMANY
Intel Semiconductor GmbH'
Seidlstrasse 77
8000 Muenchen 2
Tel 1089) 55 81 41
TELEX' 573 177

Intel *Semiconductor* GmbH
Abraham Lincoln Strasse 30
0200 Wiesbaden 1
Tel (06121) 74855
TELEX 04186183

Intel Semiconductor GmbH
D-7000 Stuttgart 83
Ernsthaldenstrasse 17
Tel (0711) 7351606
TELEX 7255346

Intel Semiconductor GmbH
Wiesenweg 26
D-6272 Niederhausen
Tel 106127) 2314

ORIENT MARKETING OFFICES

JAPAN
Intel Japan Corporation*
Flower Hill-Shinmachi East Bldg
1-23 9, Shinmachi, Setagaya-ku
Tokyo 154
Tel (03) 426-9261
TELLX. 781-28426

TAIWAN
Taiwan Automation Co *
6th Floor, 18-1, Lane 14
Chi-Lin Road
Taipei
Tel: (02) 551726-9
TELEX: 11942 TAIAUTO

INTERNATIONAL DISTRIBUTORS

ARGENTINA
S.I E S A
Av. Pte. Rogue Saenz Pena 1142 9B
1035 Buenos Aires
Tel 35-6784

AUSTRALIA
A. J. Ferguson (Ade'aide) PTY Ltd
44 Prospect Rd
Prospect 5082
South Australia 17035
Tel 269-1244
TELEX. 82635

A. J. Ferguson Electronics
34 Herbert Street
West Ryde, N.S W. 2114
Tel Ace 268-1244
TELEX 82635

Warburton-Frankie (Sydney) Pty. Ltd
199 Parramatta Road
Auburn N S.W 2114
Tel 648-1711 648-1381
TELLX WARFRAN AA 22265

Warburton-Frankie Industries
(Melbourne) Ply Ltd
220 Park Street
South Melbourne, Victoria 3205

AUSTRIA
Bacher Elektronische Gerate GmbH
Meidlinger Hauptstrasse 78
A 1120 Vienna
Tel: (0222) a3 63 96
TELEX: (01) 1532

BELGIUM
Inelco Belgium S.A.
Avenue Val Duchesse, 3
B-1160 Brussels
Tel (02) 660 00 12
TELEX. 25441

DENMARK
Scandinavian Semiconductor
Supply A/S
Nannasgade 18
DK-2200 Copenhagen N
TELEX 93 50 90
19037

FINLAND
Oy Fintronic AB
Loennrotinkatu 350
SF 00180
Helsinki 18
Tel: (90) 664 451
TELEX 12426

FRANCE
Tekelec Airtronic
Cite ces Bruyeres
Rue Carle Vernet
92310 Sevres
Tel (1) 027 75 35
TELEX: 250997

GERMANY
Alfred Neye Enatachnik GmbH
Schifferstrasse 14
D-2085 Quickborn-Hamburg
Tel. (04106) 6121
TELEX 02-13590

Electronic 2000 Vertriebs GmbH
Neumarkter Strasse 75
D-8000 Muenchen 80
Tel. (089) 434061
TELEX 522561

Jermyn GmbH
Postfach 1146
D-6277 Kamberg
Tel (06434) 6005
TELEX 486426

HONG KONG
ASTEC International
Oriental Centre
14th Floor, No. 67-71
Chatham Road
Kowloon, Hong Kong
Tel: 3-694751
Cable: "ASCOMP"
TELEX: 74899 ASCOM HX

INDIA
Electronics International
128 *Mahatma Gandhi Road*
Secunderabad
Tel 53211
TLLCX 043 222

ISRAEL
Eastronics Lld *
11 Rozanis Street
P.O. Box 39300
Tel-Avis
Tel 475151
TELEX 33638

ITALY
Eledra 3S S P A.'
Viale Elvezia, 18
20154 Milan,
Tel: (02) 3493041
TEI FX: 39332

Eledra 3S S P A *
V a Paolo Gaidano 141 D
10137 Torino
TEL (011) 30 97 097 • 3397 114

Eledra 3S S PA:
Via Giuseppe Va marana, 63
00139 Rome, Italy
Tel (06181 27 290 • 81 27 324
TELEX 65051

JAPAN
Par: Electron
No 1 Higashikata-Machi
Midori-Ku, Yokohama 226
Tel: (045) 471-8811
TELEX 781-4773

JAPAN (cont.)
Ryoyo Electric Carp
Konwa Bldg
1-12-22 Tsukiji, 1-Chome
Chuc-Ku, Tokyo 104
Tel (03) 543-7711
Nippon Micro Computer Co Ltd
Mutsumi Bldg 4-5-21 Kojimachi
Chiyoda-ku Tokyo 107
Tel (03) 230-0041

KOREA
Koram Digtal
Sam Yung Bldg #303
71-2 Bukchang Dong Chung-Ku
Seoul 100

NETHERLANDS
Inelco Nederland
AFD Elektronic
Joan Muyskenweg 22
NL-1006 Amsterdam
Tel: (020) 934824
TELEX 14322

NORWAY
Nordisk Elektronik (Norge) A/S
Mustads Vei 1
N Oslo 2
Tel (02) 55 38 93
TELEX 16963

PORTUGAL
Ditram
Componentes E Electronica LDA
Av Miquel Bombarda 133
Lisboa 1
Tel 119 45 313

SOUTH AFRICA
Electronic Building Elements
P O. Box 4609
Pretoria
Tel: 78 92 21
TELEX 30181

SPAIN
Interlace
Ronda San Pedro 22
Barcelona 10
Tel 301 7851

SWEDEN
Nordisk Electronik AB
Fack
S-10380 Stockholm 7
Tel (08) 248340
TELEX 10547

SWITZERLAND
Industrade AG
Gemsenstrasse 2
Postcheck 80 21190
CH-8021 *Zurich*
Tel (01) 60 22 30
TLLEX 56788

UNITED KINGDOM
Rapid Recall Lld
11-15 Betterton Street
Drury Lane
London WC2H 9BS
Tel (01) 379-6741
TELEX 28752

G E C Semiconductors Ltd
East Lane
Wembley HA9 7PP
Middlesex
Tel (01) 904-9303
TELEX 923429

Jermyn Industries
Vestry Estate
Sevenoaks Kenl
Tel: 10732) 50114
TELEX 95142

***Field Application** Location**

Printed in U.S.A./A226/0877/15K BL