

ADDRESS	MNEMONIC	COMMENT
	mov tmod,#01h	;set T0 to mode 1
	mov ie,#82h	;enable T0 interrupt
	setb tcon.4	;start timer
	clr psw.3	;return to register bank 0
here:	sjmp here	;loop and simulate rest of program
;		
;convert uses the PC to point to the base of the 16-byte table		
;		
convert:	inc a	;compensate for RET byte
	mov a,@pc+a	;get byte
	ret	;return with segment pattern in A
	.db c0h	;0
	.db f9h	;1
	.db a4h	;2
	.db b0h	;3
	.db 99h	;4
	.db 92h	;5
	.db 82h	;6
	.db f8h	;7
	.db f0h	;8
	.db 98h	;9
	.db 88h	;A
	.db 83h	;b
	.db c6h	;C
	.db b1h	;d
	.db 86h	;E
	.db 8eh	;F
	.end	

COMMENT

Using bank 1 as a dedicated bank for the interrupt routine cuts down on the need for pushes and pops. Bank 1 may be selected quickly, giving access to the eight registers while saving the bank 0 registers. Note that the stack, at reset, points to R0 of bank 1, so that it must be relocated.

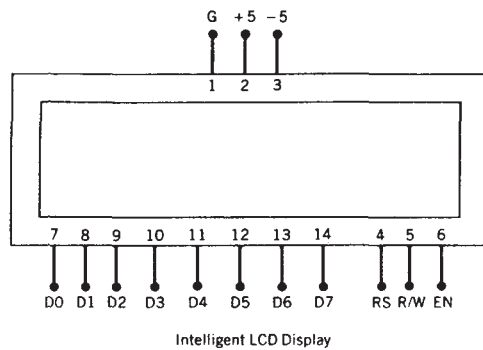
The intensity of the display may also be varied by blanking the displays completely for some interval using the program.

Intelligent LCD Display

In this section, we examine an intelligent LCD display of two lines, 20 characters per line, that is interfaced to the 8051. The protocol (handshaking) for the display is shown in Figure 8.8, and the interface to the 8051 in Figure 8.9.

The display contains two internal byte-wide registers, one for commands (RS = 0) and the second for characters to be displayed (RS = 1). It also contains a user-programmed RAM area (the character RAM) that can be programmed to generate any desired character that can be formed using a dot matrix. To distinguish between these two data areas, the hex command byte 80 will be used to signify that the display RAM address 00h is chosen.

FIGURE 8.8 Intelligent LCD Display



BIT	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Function									
	0	0	0	0	0	0	0	0	0	1	Clear LCD and memory, home cursor									
	0	0	0	0	0	0	0	0	1	0	Clear and home cursor only									
	0	0	0	0	0	0	0	1	I/O	S	Screen action as display character written S = 1/0: Shift screen/cursor									
	0	0	0	0	0	0	1	D	C	B	I/O = 1/0: Cursor R/L, screen L/R D = 1/0: Screen on/off C = 1/0: Cursor on/off B = 1/0: Cursor Blink/Noblink									
	0	0	0	0	0	1	S/C	R/L	0	0	S/C = 1/0: Screen/Cursor R/L = 1/0: Shift one space R/L									
	0	0	0	0	1	DL	N	F	0	0	DL = 1/0: 8/4 Bits per character N = 1/0: 2/1 Rows of characters F = 1/0: 5X10/5X7 Dots/Character									
	0	0	0	1	Character address						Write to character RAM Address after thr.									
	0	0	1	Display data address							Write to display RAM Address after thr.									
	0	1	BF	Current address							BF = 1/0: Busy/Notbusy									
	1	0	Character byte								Write byte to last RAM chosen									
	1	1	Character byte								Read byte from last RAM chosen									

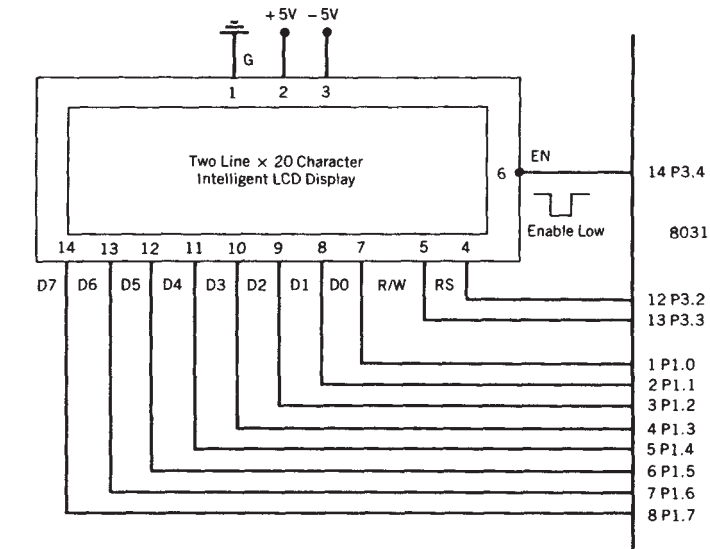
Port 1 is used to furnish the command or data byte, and ports 3.2 to 3.4 furnish register select and read/write levels.

The display takes varying amounts of time to accomplish the functions listed in Figure 8.8. LCD bit 7 is monitored for a logic high (busy) to ensure the display is not over written. A slightly more complicated LCD display (4 lines \times 40 characters) is currently being used in medical diagnostic systems to run a very similar program.

Lcdisp

The program "lcdisp" sends the message "hello" to an intelligent LCD display shown in Figure 8.8. Port 1 supplies the data byte. Port 3.2 selects the command (0) or data (1) registers. Port 3.3 enables a read (0) or write (1) level, and port 3.4 generates an active low-enable strobe.

FIGURE 8.9 Intelligent LCD Circuit for "Lcdisp" Program



ADDRESS	MNEMONIC	COMMENT
	.org 0000h	
lcdisp:	clr p3.2	;select the command register
	clr p3.3	;select write level
	mov a,#3fh	;command 8 bits/char., 2 rows, 5 x 10
	acall strobe	;strobe command to display
	mov a,#0eh	;command screen and cursor on, no blink
	acall strobe	
	mov a,#06h	;command cursor right as data displayed
	acall strobe	
	mov a,#0lh	;clear all and home cursor
	acall strobe	
	setb p3.2	;select display data RAM register
	mov a,#'h'	;say "hello"
	acall strobe	
	mov a,#'e'	
	acall strobe	
	mov a,#'l'	
	acall strobe	
	acall strobe	
	mov a,#'o'	
	acall strobe	

Continued

ADDRESS	MNEMONIC	COMMENT
<i>Continued</i>		
here:	sjmp here	;message sent
;		
;the subroutine "strobe" is used to check for a display busy		
;condition, and pulse P3.3 high-low-high to enable the display		
;write or read		
;		
strobe:	mov pl,#0ffh	;configure port 1 as an input
	setb p3.3	;set read level
wait:	setb p3.4	;generate read strobe
	clr p3.4	;enable the display
	jb pl.7,wait	;check for busy when BF = 1
	setb p3.4	;end of read strobe
	clr p3.3	;write character to display
	setb p3.2	;choose data RAM
	mov pl,a	;character to port 1
	clr p3.4	;generate write strobe
	setb p3.4	
	clr p3.2	;return with display as before call
	ret	
	.end	

COMMENT

If long character strings are to be displayed, then a subroutine could be written that receives the beginning address of the string. The subroutine then displays the characters until a unique "end-of-string" character is found.

Pulse Measurement

Sensors used for industrial and commercial control applications frequently produce pulses that contain information about the quantity sensed. Varying the sensor output frequency, using a constant duty cycle but variable frequency pulses to indicate changes in the measured variable, is most common. Varying the duration of the pulse width, resulting in constant frequency but variable duty cycle, is also used. In this section, we examine programs that deal with both techniques.

Measuring Frequency

Timers T0 and T1 can be used to measure external frequencies by configuring one timer as a counter and using the second timer to generate a timing interval over which the first can count. The frequency of the counted pulse train is then

$$\text{Unknown frequency} = \text{Counter/timer}$$

For example, if the counter counts 200 pulses over an interval of .1 second generated by the timer, the frequency is

$$UF = 200/.1 = 2000 \text{ Hz}$$

Certain fundamental limitations govern the range of frequencies that can be measured. An input pulse must make a 1-to-0 transition lasting two machine cycles, or $f/24$, to be counted. This restriction on pulse deviation yields a frequency of 667 kilohertz using our 16 megahertz crystal (assuming a square wave input).

The lowest frequency that can be counted is limited by the duration of the time interval generated, which can be exceedingly long using all the RAM to count timer rollovers ($49.15 \text{ milliseconds} \times 2^{32768}$). There is no practical limitation on the lowest frequency that can be counted.

Happily, most frequency variable sensors generate signals that fall inside of 0 to 667 kilohertz. Usually the signals have a range of 1,000 to 10,000 hertz.

Our example will use a sensor that measures dc voltage from 0 to 5 volts. At 0 V the sensor output is 1,000 hertz, and at full scale, or 5 volts, the sensor output is 6,000 hertz. The correspondence is 1 volt per 1,000 hertz, and we wish to be able to measure the voltage to the nearest .01 V, or 10 hertz of resolution (assuming the sensor is this accurate). A timing interval of 1 second generates a frequency count accurate to the nearest 1 hertz, so an interval of .1 s yields a count accurate to the nearest 10 hertz.

Another way to arrive at the desired timing interval, T , is to note that the desired accuracy is

$$\frac{.01 \text{ V}}{5 \text{ V}} = .002 = \frac{1}{512} = \frac{1}{2^9}$$

and that the range of the counter is from $T \times f_{\min}$ to $T \times f_{\max}$, or a range of $T \times (f_{\max} - f_{\min})$ from zero to full scale. The resolution of each counter bit is then

$$\text{LSB} = \frac{T \times (f_{\max} - f_{\min})}{2^n}$$

where n is the desired number of bits to be resolved. For our example, $T = 512/5000 = .1024$ seconds; .1 second yields a slightly better accuracy.

From earlier tries at generating decimal time delays in Chapter 7, it has been amply demonstrated that these cannot be done perfectly using a 16 megahertz crystal (.75 microsecond count interval). We will be close enough to meet our requirements.

T1 is used in the auto-reload mode 2 to generate overflow interrupts every 192 microseconds ($256 \times .75$ microseconds). These overflows are counted using R4 and R5 until .100032 seconds have elapsed (521d overflows). For this example, T0 is used as a counter to count the external frequency that is fed to the port 3.4 (T0) pin during the T1 interval. Using the interval chosen, the range of counts in T0 becomes

$$\begin{aligned} 0\text{V} &= 1000 \text{ Hz} \times .100032 \text{ s} = 100\text{d counts} \\ 5\text{V} &= 6000 \text{ Hz} \times .100032 \text{ s} = 600\text{d counts} \\ .01\text{V} &= 10\text{Hz} \times .100032 \text{ s} = 1 \text{ count} \end{aligned}$$

which meets the desired accuracy specification.

Freq

The program "freq" uses T0 to count an external pulse train that is known to vary in frequency from 1000 to 6000 hertz. T1 generates an exact time delay of 192 microseconds that is counted using registers R4 and R5 of bank 1 until T1 has overflowed 521d times, or a total delay of .100032 seconds.

ADDRESS	MNEMONIC	COMMENT
	.equ frqflg,0fh	;use addressable bit for a flag
	.org 0000h	
freq:	mov sp,#0fh	;set stack above register bank one
	sjmp over	;jump over the T1 interrupt location
;		
;T1 will overflow and vector here: R4 and R5 will be used as a		
;combined 16-bit counter to count the 521d overflows; the extra		
;microseconds needed to detect end of count and stop T0 will		
;introduce a slight error		
;		
	.org 001bh	;place program at T1 interrupt vector
	setb psw.3	;switch to register bank 1
	djnz r4,timup	;count R4 down and test for 521d
		;counts
timup:	dec r5	
	cjne r5,#0fdh,go	;count down from 0000 to FDF7h (209h)
	cjne r4,#0f7h,go	;209h = 521d
	clr tcon.4	;stop T0 and set frqflg
	setb frqflg	;main program can now process
		;frequency
	clr tcon.6	;stop T1
go:	clr psw.3	;return to register bank zero
	reti	;total extra time to stop T0 =
		;8.25 μ s
;		
;the main program sets up T0 to be a counter and starts T1; the		
;flag frqflg is then watched until it is set by the interrupt		
;program; the main program must do this every time a frequency read		
;is desired; if continuous frequency determinations are desired by		
;the main program, then the interrupt program could call a subroutine		
;frequency handling program inserted before "go" in place of the		
;instruction that stops T1.		
;		
over:	setb psw.3	;select register bank one
	mov r4,#00h	;reset R4 and R5
	mov r5,#00h	
	clr psw.3	;restore to register bank zero
	mov tmod,#25h	;T1 mode 2 timer, T0 mode 1 counter
	mov t1l,#00h	;count up from 00 and reset
	mov t1h,#00h	;reload with 00
	mov tcon,#50h	;start T0 and T1
	mov ie,#88h	;enable T1 to interrupt
simulate:	jbc frqflg,getfrq	;simulate main program getting data
	sjmp simulate	
getfrq:	nop	;place frequency subroutine here
	sjmp simulate	
	.end	

COMMENT

The longer the time taken to count, the more accurate the frequency will be (but remember, it makes little sense to make the readout more accurate than the basic sensor). T0 will overflow at 65,535 or at the end of an interval of 10.92 s at f_{\max} , which can be generated in T1 and R4, R5. In this case, the accuracy would be to the nearest .09 hertz (.0001 volt).

If you wish to generate a delay closer to .1 s than used in the example, make T1 cycle in a shorter period of time and count these shorter periods in R4, R5. Compensate for the 8.5 microseconds it takes for the interrupt routine to determine that time is up.

Preloading T0 with a number that causes T0 to overflow to 0000 when f_{\min} is present during T will enable T0 to read the voltage directly. For our example, presetting T0 to FF9Ch will have $T0 = 01F4h$ (500d) at $f_{\max} = 60,000$ hertz for $T = .1$ s.

Pulse Width Measurement

Theoretically, if the input pulse is known to be a perfect square wave, the pulse frequency can be measured by finding the time the wave is high (T_h). The frequency is then

$$UF = \frac{1}{T_h \times 2}$$

If T_h is 200 microseconds, for example, then UF is 2500 hertz. The accuracy of the measurement will fall as the input wave departs from a 50 percent duty cycle.

Timer X may be configured so that the internal clock is counted only when the corresponding \overline{INTX} pin is high by setting the GATE X bit in TMOD. The accuracy of the measurement is within approximately one-half of the timer clock period, or .375 microsecond for a 16 megahertz crystal. This accuracy can only be attained if the measurement is started when the input wave is low and stopped when the input next goes low. Pulse widths greater than the capacity of the counter, which is 49,152 milliseconds for a 16 megahertz crystal, can be measured by counting the overflows of the timer flag and adding the final contents in the counter.

For the example in this section, the sensor used to measure the 0 volt to 5 volts dc voltage has a fixed frequency of 1000 hertz or a period of 1 ms. For a 0 volt input, the sensor is high for 400 microseconds and low for 600 microseconds; when the sensor input is 5 volts, the output is high for 900 microseconds and low for 100 microseconds. Each volt represents 100 microseconds of time; the accuracy of the measurement is $\pm .00325$ volts, which is within the specification of .01 volt.

To make the measurement, T0 will be configured to count the internal clock when $\overline{INT0}$ is high. The measurement is not started until $\overline{INT0}$ goes from high to low, leaving a minimum of 100 microseconds to start T0. The measurement is made while $\overline{INT0}$ is high and stopped when $\overline{INT0}$ goes low again. The whole process can be interrupt driven by using the interrupt flag associated with $\overline{INT0}$. The IE0 flag can be set whenever $\overline{INT0}$ goes from high to low to notify the program to start the pulse width timing and then to stop. A variation of this program is currently in use to measure fabric width by measuring the reflection time of a scanning laser.

Width

The program "Width" measures the width of pulses that are fed to the $\overline{INT0}$ pin, port 3.2 and that are known to vary from 400 to 900 microseconds. The program starts when the interrupt flag IE0 is set and stops the next time the flag is set, indicating one complete cycle of the input wave.

ADDRESS	MNEMONIC	COMMENT
	.equ wflg,00h	;flag set to notify main program
	.org 0000h	
width:	sjump over	;jump over $\overline{\text{INT0}}$ flag vector location
	;the $\overline{\text{INT0}}$ edge triggered flag will vector here	
	.org 0003h	
	jbc tcon.4,stop	;if T0 is running, stop T0
	setb tcon.4	;if T0 is not running, enable T0
	clr wflg	;reset wflg until next measurement
	reti	;return with T0 enabled
stop:	setb wflg	;set flag for main program
	reti	;return with T0 stopped
	;the main program resumes here; the program monitors the flag that	
	;indicates that a width measurement has just been made	
over:	;mov tmod,#09h	;set T0 to count when $\overline{\text{INT0}}$ high
	mov tcon,#01h	;enable edge trigger for $\overline{\text{INT0}}$
	mov tl0,#00h	;reset T0
	mov th0,#00h	
	mov ie,#81h	;enable external interrupt
simulate:	jbc wflg,getw	;look for wflg and get width
	sjmp simulate	
getw:	nop	;real program would read T0 for width
	mov tl0,#00h	;reset T0
	mov th0,#00h	
	sjmp simulate	;simulate main program
	.end	

COMMENT

If there is a considerable amount of electrical noise present on the $\overline{\text{INT0}}$ pin, an average value of the pulse width could be found by measuring the widths of a number of consecutive pulses. A counter could be incremented at the end of each cycle and the sum of the widths divided by the counter contents. The noise should average to zero.

Frequency can be measured by timing the interval of a number (M) of high-to-low $\overline{\text{INTX}}$ interrupts. Synchronize the timing by starting the timer at the first transition, and stop the timer at the Mth + 1 transition. The frequency is then

$$UF = \frac{M}{T}$$

where T is the count in the timer.

D/A and A/D Conversions

Conversion between the analog and digital worlds requires the use of integrated circuits that have been designed to interface with computers. Highly intelligent converters are commercially available that all have the following essential characteristics:

Parallel data bus: tri-state, 8-bit

Control bus: enable (chip select), read/write, ready/busy

The choice the designer must make is whether to use the converter as a RAM memory location connected to the memory busses or as an I/O device connected to the ports. Once that choice is made, the set of instructions available to the programmer becomes limited. The memory location assignment is the most restrictive, having only MOVX available. The design could use the additional 32K RAM address space with the addition of circuitry for A15. By enabling the RAM when A15 is low, and the converter when A15 is high, the designer could use the upper 32K RAM address space for the converter, as was done to expand port capacity by memory mapping in Chapter 7. All of the examples examined here are connected to the ports.

D/A Conversions

A generic R-2R type D/A converter, based on several commercial models, is connected to ports 1 and 3 as shown in Figure 8.10. Port 1 furnishes the digital byte to be converted to an analog voltage; port 3 controls the conversion process. The converter has these features:

$$V_{out} = -V_{ref} \times (\text{byte in}/100H), V_{ref} = \pm 10 \text{ V}$$

Conversion time: $5 \mu s$

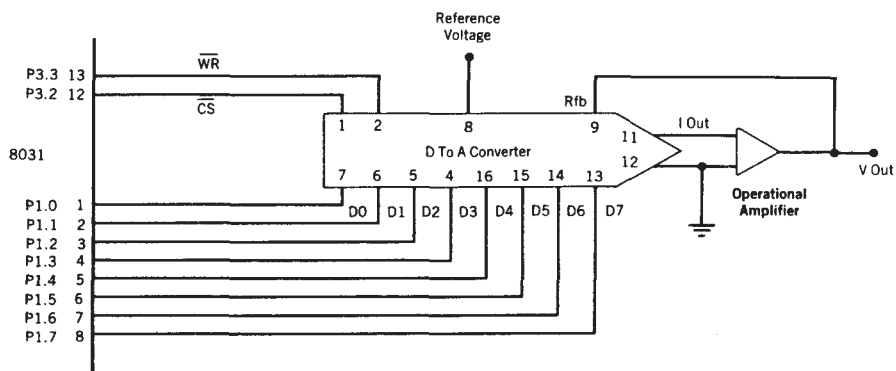
Control sequence: \overline{CS} then \overline{WR}

For this example, a 1000 hertz sine wave that will be generated can have a programmable frequency. V_{ref} is chosen to be -10 volts, and the wave will swing from $+9.96$ volts to 0 volt around a midpoint of 4.48 volts. The program uses a lookup table to generate the amplitude of each point of the sine wave; the time interval at which the converter is fed bytes from the table determines the wave frequency.

The conversion time limits the highest frequency that can be generated using S sample point. In this example, the shortest period that can be used is

$$T_{min} = S \times 5 \mu s = 5S \mu s, \quad f_{max} = \frac{200,000}{S}$$

FIGURE 8.10 D/A Converter Circuit for "Davcon" Program



The design tension is high frequency versus high resolution. For a 1000 hertz wave, S could be 2000 samples. In reality, we cannot use this many samples; the program cannot fetch the data, latch it to port 1, and strobe port 3.3 in 5 microseconds. An inspection of the program will show that the time needed for a single wave point is 6 microseconds, and setting up for the next wave takes another 2.25 microseconds. S becomes 166d samples using the 6 microseconds interval, and the addition of 2.25 microseconds at the end of every wave yields a true frequency of 1001.75 hertz.

Davcon

The D/A converter program "Davcon" generates a 1000 hertz sine wave using an 8-bit converter. 166d samples are stored in a lookup table and fed to the converter at a rate of one sample every 6 microseconds. The lookup table is pointed to in external ROM by the DPTR, and R1 is used to count the samples. Numbers in parentheses indicate the number of cycles.

ADDRESS	MNEMONIC	COMMENT
	.org 0000h	
davcon:	clr p3.2	;enable chip select to converter
	mov dptr,#table	;get base address to DPTR
repeat:	mov r1,#0a6h	;initialize R1 to 166d (1)
next:	mov a,r1	;offset into table (1)
	movc a,@a+dptr	;get sample (2)
	mov pl.a	;sample to port 1 (1)
	clr p3.3	;write strobe low (1)
	setb p3.3	;write strobe high (1)
	djnz r1,next	;loop for 166D samples (2)
	sjmp repeat	;reload R1 and generate next wave (2)
;		
;the lookup table begins here; a cosine wave is chosen to make the		
;table readable; the first 83 samples cover the wave from maximum to		
;1 less than 0; the next 83 cover the wave from 0 to maximum. 83		
;samples per half-cycle means a sample every 2.17 degrees		
;		
table:	.db 00h	;no entry at A = 00h
	.db ffh	;FFhcos 0 = FFh. s1
	.db feh	;7Fh + 7Fhcos 2.17 = FEh. s2
	.db feh	;7Fh + 7Fhcos 4.34 = FEh. s3
	.db fdh	;sample 4
	.db fdh	;sample 5
	;and so on until we near 90 degrees:	
	.db 81h	;7Fh + 7Fhcos 88.9 = 81h. s42
	.db 7ch	;7Fh + 7Fhcos 91.1 = 7Ch. s43
	;near 180 degrees we have:	
	.db 01h	;7Fh + 7Fh cos 173.5 = 01h. s81
	.db 00h	;7Fh + 7Fh cos 175.7 = 00h. s82
	.db 00h	;7Fh + 7Fh cos 177.8 = 00h. s83
	.db 00h	;7Fh + 7Fh cos 180 = 00h. s84.
	.db 00h	;7Fh + 7Fh cos 182.2 = 00h. s85
	.db 00h	;7Fh + 7Fh cos 184.33 = 00h. s86

Continued

ADDRESS	MNEMONIC	COMMENT
	.db 01h	:7Fh + 7Fh cos 186.5 = 01h. s87
;finally, close to 360 degrees the table contains:		
	.db fbh	;s 161
	.db fch	;s 162
	.db fdh	;s 163
	.db fdh	;s 164
	.db feh	;s 165
	.db feh	;s 166
	.end	

COMMENT

The program retrieves the data from the highest to the lowest address.

A/D Conversion

The easiest A/D converters to use are the "flash" types, which make conversions based upon an array of internal comparators. The conversion is very fast, typically in less than 1 microsecond. Thus, the converter can be told to start, and the digital equivalent of the input analog value will be read one or two instructions later. Modern successive approximation register (SAR) converters do not lag far behind, however, with conversion times in the 2–4 microsecond range for eight bits.

At this writing, flash converters are more expensive (by a factor of two) than the traditional SAR types, but this cost differential should disappear within four years. Typical features of an eight-bit flash converter are

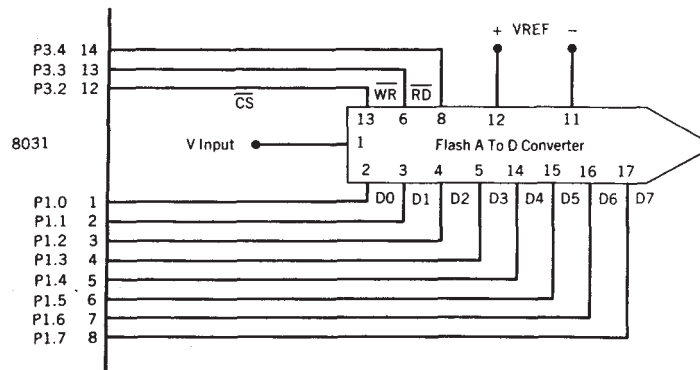
Data: $V_{in} = V_{ref}(-)$, data = 00h; $V_{in} = V_{ref}(+)$, data = FFh

Conversion time: 1 μ s

Control sequence: \overline{CS} then \overline{WR} then \overline{RD}

An example circuit, using a generic flash converter, is shown in Figure 8.11. Port 1 is used to read the byte value of the input analog voltage, and port 3 controls the conversion.

FIGURE 8.11 A/D Converter Circuit for "Adconv" Program



A conversion is started by pulsing the write line low, and the data is read by bringing the read line low.

Our example involves the digitizing of an input waveform every 100d microseconds until 1000d samples have been stored in external RAM.

Adconv

The program "Adconv" will digitize an input voltage by sampling the input every 100 μ s and storing the digitized values in external RAM locations 4000h to 43E7h (1000d samples). Numbers in parentheses are cycles. The actual delay between samples is 99.75 microseconds.

ADDRESS	MNEMONIC	COMMENT
	.equ begin,4000h	;start storage at 4000h
	.equ delay,74h	;delay in DJNZ loop for 87 usec
	.equ endl,43h	;high byte of ending address
	.equ end2,e8h	;low byte of ending address
	.org 0000h	
adconv:	mov dptr,#begin	;point to starting address in RAM
	clr p3.2	;generate \overline{CS} to ADC
next:	clr p3.3	;generate \overline{WR} pulse (1)
	setb p3.3	;(1)
	clr p3.4	;generate \overline{RD} pulse (1)
	mov a,pl	;get data (1)
	setb p3.4	;end of \overline{RD} pulse (1)
	movx @dptr,a	;store in external RAM (2)
	inc dptr	;point to next and see if done (2)
	mov a,dph	;(1)
	cjne a,#endl,wait	;(2)
	mov a,dpl	;(1)
	cjne a,#end2,wait	;(2)
	sjmp done	;finished if both tests pass
wait:	mov rl,#delay	;delay for 87d μ s
here:	djnz rl,here	;(2) $\times .75 \mu$ s $\times 116d = 87 \mu$ s
	sjmp next	;(2) 17d cycles (12.75 μ s)
done:	sjmp done	;simulate rest of program
	.end	

COMMENT

Using this program, we could fill up the RAM in 3.2 s, which illustrates the volumes of data that can be gathered quickly by such a circuit. Realistic applications would feature some data reduction at the microcontroller before the reduced (massaged) data were relayed to a host computer.

Multiple Interrupts

The 8051 is equipped with two external interrupt input pins: $\overline{INT0}$ and $\overline{INT1}$ (P3.2 and P3.3). These are sufficient for small systems, but the need may arise for more than two interrupt points. There are many schemes available to multiply the number of interrupt points; they all depend upon the following strategies: