THE

PRIMER

TRAINER

APPLICATION MANUAL

Manual Revision 2.0

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL

CARBONDALE, IL 62901
618-529-4525

Table of Contents

Application 1: Count Down Timer

This program will count down from the packed BCD number in the HL register pair to 0 at a time increment determined by the hex number in the DE register pair. When the count = 0, the alarm will sound and the LEDs will light. The alarm can be discontinued and the program terminated by pressing any key on the keypad. After typing in the program, load the HL and DE register pairs as follows:

> Load the HL register pair with the desired time interval.
> Format = packed BCD range = 9999 to 0001
>
> Load DE register pair with the time scaler.
> Format = hex range = 0001h to FFFFh

The time scaler determines how many hundredths of seconds must pass before the counter is decremented. The time interval between decrements will be ((time scaler) / 100) seconds. For example, if the scaler is 0064h (100 decimal) the timer will decrement once a second. If the scaler is 1770h (6000 decimal) the timer will count decrement once every 60 seconds.

The assembly language code is listed below:

```
                ; ---------------------------------------------------------;
                ; ...........EQUATES.....................................;
                ; ---------------------------------------------------------;

FFE9    =       VEC7HLF:    EQU     0FFE9H       ; INT 7.5 VECTOR
0000    =       SCALELO:    EQU     00H          ; 307200HZ / 768 =
004C    =       SCALEHI:    EQU     4CH          ; 100HZ TICK RATE
0014    =       TIMERLO:    EQU     14H          ; TIMER PORTS
0015    =       TIMERHI:    EQU     15H
00CD    =       TIMCMD:     EQU     0CDH         ; TIMER FUNC. COMMAND
0010    =       CMDREG:     EQU     10H          ; TIMER COMMAND PORT
001A    =       INTMASK:    EQU     1AH          ; INTERRUPT MASK
FF01    =       TIMPROG:    EQU     0FF01H       ; RTC PROG START ADDR
000C    =       SERVC:      EQU     0CH          ; EMOS SERVICES
0012    =       SERV12:     EQU     12H
000B    =       SERV0B:     EQU     0BH
1000    =       MOS:        EQU     1000H        ; MOS CALL LOCATION
00FF    =       LIGHT:      EQU     0FFH         ; ALARM LED ON PATTERN
0000    =       DARK:       EQU     0            ; ALARM LED OFF PATTERN


                ; ---------------------------------------------------------;
FF01    ORG     TIMPROG
                ; ---------------------------------------------------------;


                ; ---------------------------------------------------------;
                ; ...........INITIALIZE..................................;
                ; ---------------------------------------------------------;

FF01    F3      START:      DI                   ; DISABLE INTERRUPTS
FF02    22AEFF              SHLD    TIM1         ; LOAD H/L TO TIMER1
FF05    EB                  XCHG
FF06    22A4FF              SHLD    SCALER       ; D/E CONTAINS SCALER
FF09    2157FF              LXI     H,TIMERS     ; ON 7.5 INTERRUPT
```

```
FF0C    22E9FF                  SHLD    VEC7HLF     ; VECTOR TO RTC
FF0F    3E00                    MVI     A,SCALELO   ; SET LOW COUNT BYTE
FF11    D314      OUT           TIMERLO             ; OF TIMER CHIP
FF13    3E4C      MVI           A,SCALEHI           ; SET HIGH COUN T BYTE
FF15    D315      OUT           TIMERHI             ; OF TIMER CHIP
FF17    3ECD      MVI           A,TIMCMD            ; SET TIMER CHIP FOR
FF19    D310      OUT           CMDREG              ; 100 HZ SQUARE WAVE
FF1B    3E01      MVI           A,01H               ; SET ALARM FLAG TO
FF1D    32B0FF    STA           ALRMFLAG            ; ARM ALARM
FF20    2AA4FF    LHLD          SCALER              ; INITIALIZE TIMER 0
FF23    22ACFF    SHLD          TIM0
FF26    3E1A      MVI           A,INTMASK           ; UNMASK 7.5 AND 5.5
FF28    30        SIM                               ; INTERRUPTS
FF29    FB        EI                                ; ENABLE INTERRUPTS

                  ; ----------------------------------------------------------;
                  ; ...........MAIN PROGAM...................................;
                  ; ----------------------------------------------------------;

FF2A    0E12      DOTIME:       MVI     C,SERV12    ; USE SERVICE 12
FF2C    2AAEFF                  LHLD    TIM1        ; TO DISPLAY TIMER 1
FF2F    EB                      XCHG                ; DE WILL BE DISPLAYED
FF30    CD0010                  CALL    MOS         ; CALL MOS
FF33    3AB0FF                  LDA     ALRMFLAG    ; IF ALARM IS ON
FF36    FE01                    CPI     01H         ; GO WAIT FOR KEY
FF38    CA2AFF                  JZ      DOTIME      ; ELSE DISPLAY TIMER
FF3B    0E12                    MVI     C,SERV12    ; MAKE SURE WE DISPLAY
FF3D    2AAEFF                  LHLD    TIM1        ; ONE LAST TIME TO
FF40    EB                      XCHG                ; DISPLAY TERMINAL
FF41    CD0010                  CALL    MOS         ; COUNT
FF44    0E0B                    MVI     C,SERV0B    ; STRIKE ANY KEY
FF46    CD0010                  CALL    MOS         ; TO STOP ALARM
FF49    20                      RIM                 ; SPEAKER OFF
FF4A    F640                    ORI     40H
FF4C    E67F                    ANI     7FH
FF4E    30                      SIM
FF4F    0E0C                    MVI     C,SERVC     ; LEDS OFF
FF51    1E00                    MVI     E,DARK
FF53    CD0010                  CALL    MOS
FF56    FF                      RST     7           ; RETURN TO MOS

                  ; ----------------------------------------------------------;
                  ; ...........7.5 INTERRUPT HANDLER.........................;
                  ; ----------------------------------------------------------;

FF57    F5        TIMERS:       PUSH    PSW
FF58    E5                      PUSH    H
FF59    2AACFF                  LHLD    TIM0        ; GET TIM0
FF5C    7D                      MOV     A,L         ; IF ITS NOT ZERO
FF5D    B4                      ORA     H
FF5E    C29CFF                  JNZ     DECTIM0     ; DECREMENT TIM0
FF61    2AA4FF                  LHLD    SCALER      ; ELSE TIM0 = 100
FF64    22ACFF                  SHLD    TIM0        ; RELOAD TIM0
FF67    3AAEFF                  LDA     TIM1        ; GET TIM1 LOW
FF6A    C699                    ADI     99H         ; DECREMENT
FF6C    27                      DAA                 ; DECIMAL ADJUST
FF6D    32AEFF                  STA     TIM1        ; STORE TIM1 LOW
```

```
FF70   3AAFFF                LDA     TIM1+01H     ; GET TIM1 HIGH
FF73   CE99                  ACI     99H          ; DECREMENT
FF75   27                    DAA                  ; DECIMAL ADJUST
FF76   32AFFF                STA     TIM1+01H     ; STORE TIM1 HIGH
FF79   2AAEFF                LHLD    TIM1         ; GET TIM1
FF7C   7D                    MOV     A,L          ; IF ITS NOT ZERO
FF7D   B4                    ORA     H
FF7E   C2A0FF                JNZ     EXITTIME     ; EXIT
FF81   3AB0FF                LDA     ALRMFLAG     ; IF ALARM HAS
FF84   FE00                  CPI     00H          ; BEEN ACTIVATED
FF86   CAA0FF                JZ      EXITTIME     ; EXIT
FF89   3E00                  MVI     A,00H        ; ELSE, ZERO ALARM
FF8B   32B0FF                STA     ALRMFLAG     ; FLAG & ACTIVATE
FF8E   20                    RIM                  ; SPEAKER ON
FF8F   F6C0                  ORI     0C0H
FF91   30                    SIM
FF92   0E0C                  MVI     C,SERVC      ; LEDS ON
FF94   1EFF                  MVI     E,LIGHT
FF96   CD0010                CALL    MOS
FF99   C3A0FF                JMP     EXITTIME     ; EXIT
FF9C   2B       DECTIM0:     DCX     H            ; DECREMENT TIM0
FF9D   22ACFF                SHLD    TIM0
FFA0   E1       EXITTIME:    POP     H            ; RECOVER REGISTERS
FFA1   F1                    POP     PSW
FFA2   FB                    EI
FFA3   C9                    RET                  ; RETURN

                ; -------------------------------------------------------;
                ; ...........SUBROUTINES.................................;
                ; -------------------------------------------------------;

                ; -------------------------------------------------------;
                ; ...........DATA STORAGE................................;
                ; -------------------------------------------------------;

FFA4            SCALER:      DS      02H          ; DETERMINES TIME INCR.
FFA6            DISPBUFF:    DS      06H          ; DISPLAY BUFFER
FFAC            TIM0:        DS      02H
FFAE            TIM1:        DS      02H          ; SOFTWARE TIMER 1
FFB0            ALRMFLAG:    DS      01H          ; ALARM FLAG.0 = NO ALRM
                ; -------------------------------------------------------;
FFB1            END
```

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION
FF01	F3	DI
FF02	22	SHLD FFAE
FF03	AE	
FF04	FF	
FF05	EB	XCHG
FF06	22	SHLD FFA4
FF07	A4	
FF08	FF	
FF09	21	LXI H,FF57
FF0A	57	
FF0B	FF	

ADDRESS	DATA	DESCRIPTION
FF0C	22	SHLD FFE9
FF0D	E9	
FF0E	FF	
FF0F	3E	MVI A,00
FF10	00	
FF11	D3	OUT 14
FF12	14	
FF13	3E	MVI A,4C
FF14	4C	

Continued on next page…

ADDRESS	DATA	DESCRIPTION	
FF15	D3	OUT	15
FF16	15		
FF17	3E	MVI	A,CD
FF18	CD		
FF19	D3	OUT	10
FF1A	10		
FF1B	3E	MVI	A,01
FF1C	01		
FF1D	32	STA	FFB0
FF1E	B0		
FF1F	FF		
FF20	2A	LHLD	FFA4
FF21	A4		
FF22	FF		
FF23	22	SHLD	FFAC
FF24	AC		
FF25	FF		
FF26	3E	MVI	A,1A
FF27	1A		
FF28	30	SIM	
FF29	FB	EI	
FF2A	0E	MVI	C,12
FF2B	12		
FF2C	2A	LHLD	FFAE
FF2D	AE		
FF2E	FF		
FF2F	EB	XCHG	
FF30	CD	CALL	1000
FF31	00		
FF32	10		
FF33	3A	LDA	FFB0
FF34	B0		
FF35	FF		
FF36	FE	CPI	01
FF37	01		
FF38	CA	JZ	FF2A
FF39	2A		
FF3A	FF		
FF3B	0E	MVI	C,12
FF3C	12		
FF3D	2A	LHLD	FFAE
FF3E	AE		
FF3F	FF		
FF40	EB	XCHG	
FF41	CD	CALL	1000
FF42	00		
FF43	10		
FF44	0E	MVI	C,0B
FF45	0B		
FF46	CD	CALL	1000
FF47	00		
FF48	10		
FF49	20	RIM	
FF4A	F6	ORI	40
FF4B	40		

ADDRESS	DATA	DESCRIPTION	
FF4C	E6	ANI	7F
FF4D	7F		
FF4E	30	SIM	
FF4F	0E	MVI	C,0C
FF50	0C		
FF51	1E	MVI	E,00
FF52	00		
FF53	CD	CALL	1000
FF54	00		
FF55	10		
FF56	FF	RST	7
FF57	F5	PUSH	PSW
FF58	E5	PUSH	H
FF59	2A	LHLD	FFAC
FF5A	AC		
FF5B	FF		
FF5C	7D	MOV	A,L
FF5D	B4	ORA	H
FF5E	C2	JNZ	FF9C
FF5F	9C		
FF60	FF		
FF61	2A	LHLD	FFA4
FF62	A4		
FF63	FF		
FF64	22	SHLD	FFAC
FF65	AC		
FF66	FF		
FF67	3A	LDA	FFAE
FF68	AE		
FF69	FF		
FF6A	C6	ADI	99
FF6B	99		
FF6C	27	DAA	
FF6D	32	STA	FFAE
FF6E	AE		
FF6F	FF		
FF70	3A	LDA	FFAF
FF71	AF		
FF72	FF		
FF73	CE	ACI	99
FF74	99		
FF75	27	DAA	
FF76	32	STA	FFAF
FF77	AF		
FF78	FF		
FF79	2A	LHLD	FFAE
FF7A	AE		
FF7B	FF		
FF7C	7D	MOV	A,L
FF7D	B4	ORA	H
FF7E	C2	JNZ	FFA0
FF7F	A0		
FF80	FF		

Continued on next page…

ADDRESS	DATA	DESCRIPTION	
FF81	3A	LDA	FFB0
FF82	B0		
FF83	FF		
FF84	FE	CPI	00
FF85	00		
FF86	CA	JZ	FFA0
FF87	A0		
FF88	FF		
FF89	3E	MVI	A,00
FF8A	00		
FF8B	32	STA	FFB0
FF8C	B0		
FF8D	FF		
FF8E	20	RIM	
FF8F	F6	ORI	C0
FF90	C0		
FF91	30	SIM	
FF92	0E	MVI	C,0C

ADDRESS	DATA	DESCRIPTION	
FF93	0C		
FF94	1E	MVI	E,FF
FF95	FF		
FF96	CD	CALL	1000
FF97	00		
FF98	10		
FF99	C3	JMP	FFA0
FF9A	A0		
FF9B	FF		
FF9C	2B	DCX	H
FF9D	22	SHLD	FFAC
FF9E	AC		
FF9F	FF		
FFA0	E1	POP	H
FFA1	F1	POP	PSW
FFA2	FB	EI	
FFA3	C9	RET	

Application 2: Waveform Generator

This application allows the user to output 4 different waveforms (sine, square, triangle and sawtooth) from the digital to analog converter. The desired waveform can be selected by moving DIP switches 6 and 7 to one of 4 possible combinations. The frequency of the waveforms can be changed by moving DIP switches 0 through 5.

The assembly language code is listed below:

```
timerhi: equ      15h          ; the timer mode and MSB of count length
timerlo: equ      14h          ; the LSB of count length
dip:     equ      12h          ; DIP switch port
dacout:  equ      13h          ; Digital to analog output port
cmdreg:  equ      10h          ; 8155 control register.

         org      0ff01h
getime:  in       dip          ; get value of DIP switches
         add      a            ; shift left padding zeros
         add      a            ; shift left padding zeros
         out      timerlo      ; set the low count
         mvi      a,11000000b
         out      timerhi      ; single pulse w/auto reload
         mvi      a,0cdh
         out      cmdreg       ; enable timer

         in       dip          ; read DIP again
         ani      11000000b    ; Mask all DIP bits except 6 and 7
         cpi      0
         jz       sinewv       ; if upper bits are 0, output sine wave
         cpi      01000000b
         jz       sqrwav       ; if upper 2 bits are 01, output square wave
         cpi      10000000b
         jz       triang       ; if upper 2 bits are 10, output triangle wave

         ; If none of the above, upper 2 bits are 11, so output a .......
         ; sawtooth wave
sawwav:  mvi      c,0          ; invert the pattern
         mvi      d,3fh        ; starting value to output
         jmp      trian2

         ; triangle wave
triang:  mvi      c,1
         mvi      d,0          ; upward slope 0 to 3e
trian1:  mov      a,d
         call     dactim       ; output the pattern to DAC and wait
         inr      d
         mvi      a,3fh        ; if D = 3F then slope down
         cmp      d
         jnz      trian1

trian2:  mov      a,d          ; downward slope 3f to 1
         call     dactim       ; output the pattern to DAC and wait
         dcr      d
         jnz      trian2
         jmp      getime       ; check DIP switch
```

```
        ; square wave
sqrwav: mvi     c,1         ; non-inverted output
sqrwv2: mvi     d,32        ; output 32 times for each half of period
sqrwv3: xra     a
        call    dactim      ; output the pattern to DAC and wait
        dcr     d
        jnz     sqrwv3      ; jump if not output 32 times already
        dcr     c           ; change to inverted output mode
        jz      sqrwv2      ; if c=0 then sqrwv2
        jmp     getime      ; c=FF so check DIP switch

        ; sine wave
sinewv: lxi     h,sintbl    ; point to sine table
quadst: mvi     c,1         ; C=1 = 1st 2 quadrants, C=0 2nd two quadrants
quad1:  inx     h           ; skip the 0
qud1lp: inx     h
        mov     a,m         ; A is value from table
        ora     a           ; set Z flag if A = 0
        jz      quad2       ; if A = 0 then read the table backwards
        call    dactim      ; output the pattern to DAC and wait
        jmp     qud1lp

quad2:  dcx     h           ; skip the 0
qud2lp: dcx     h
        mov     a,m         ; A is value from table
        ora     a           ; set Z flag if A = 0
        jz      quad3       ; if A=0 then invert the output pattern
        call    dactim      ; output the pattern to DAC and wait
        jmp     qud2lp

quad3:  dcr     c           ; change invert flag
        jz      quad1       ; if C=0 start over but invert data
        jmp     getime      ; if C=FF then check DIP switch

        ; DACTIM: This subroutine examines the C register and if C=0
        ; it will invert the data in the A register otherwise if C=1 it
        ; will not. The A register is then output to the D to A convertor.
        ; After this, the RST 7.5 interrupt flag will be polled until a
        ; pulse is sent from the 8155 timer. This causes the program to
        ; pause after each output from the D to A convertor according to
        ; the length of the timer count.
dactim: inr     c           ; see what C is .... (0 or 1)
        dcr     c           ; ...without changing it
        jnz     dactim1     ; jump if C = 1 and don't invert data
        mov     b,a         ; invert the data
        mvi     a,3fh       ; by subtracting it from this value
        sub     b

dactim1: out    dacout      ; output the data
polltmr: rim    ; loop until rst 7.5 flag is high
        ani     01000000b   ; mask all but rst 7.5 flag
        jz      polltmr     ; check it again if not set
        mvi     a,10h
        sim     ; clear the interrupt flag
        ret
```

```
        ; This is 1 quadrant of the sine wave pattern with zeros marking
        ; the start and the end.
sintbl: defb     0, 1Fh,21h,23h,25h, 27h,29h,2Bh,2Dh, 2Eh,30h,32h,34h, 35h
        defb     36h,38h,39h,3Ah, 3Bh,3Ch,3Dh,3Dh, 3Eh,3Eh,3Fh,3Fh, 3Fh, 0
        end
```

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION	
FF01	DB	IN	12
FF02	12		
FF03	87	ADD	A
FF04	87	ADD	A
FF05	D3	OUT	14
FF06	14		
FF07	3E	MVI	A,C0
FF08	C0		
FF09	D3	OUT	15
FF0A	15		
FF0B	3E	MVI	A,CD
FF0C	CD		
FF0D	D3	OUT	10
FF0E	10		
FF0F	DB	IN	12
FF10	12		
FF11	E6	ANI	C0
FF12	C0		
FF13	FE	CPI	00
FF14	00		
FF15	CA	JZ	FF56
FF16	56		
FF17	FF		
FF18	FE	CPI	40
FF19	40		
FF1A	CA	JZ	FF43
FF1B	43		
FF1C	FF		
FF1D	FE	CPI	80
FF1E	80		
FF1F	CA	JZ	FF29
FF20	29		
FF21	FF		
FF22	0E	MVI	C,00
FF23	00		
FF24	16	MVI	D,3F
FF25	3F		
FF26	C3	JMP	FF38
FF27	38		
FF28	FF		
FF29	0E	MVI	C,01
FF2A	01		
FF2B	16	MVI	D,00
FF2C	00		
FF2D	7A	MOV	A,D
FF2E	CD	CALL	FF7C
FF2F	7C		
FF30	FF		

ADDRESS	DATA	DESCRIPTION	
FF31	14	INR	D
FF32	3E	MVI	A,3F
FF33	3F		
FF34	BA	CMP	D
FF35	C2	JNZ	FF2D
FF36	2D		
FF37	FF		
FF38	7A	MOV	A,D
FF39	CD	CALL	FF7C
FF3A	7C		
FF3B	FF		
FF3C	15	DCR	D
FF3D	C2	JNZ	FF38
FF3E	38		
FF3F	FF		
FF40	C3	JMP	FF01
FF41	01		
FF42	FF		
FF43	0E	MVI	C,01
FF44	01		
FF45	16	MVI	D,20
FF46	20		
FF47	AF	XRA	A
FF48	CD	CALL	FF7C
FF49	7C		
FF4A	FF		
FF4B	15	DCR	D
FF4C	C2	JNZ	FF47
FF4D	47		
FF4E	FF		
FF4F	0D	DCR	C
FF50	CA	JZ	FF45
FF51	45		
FF52	FF		
FF53	C3	JMP	FF01
FF54	01		
FF55	FF		
FF56	21	LXI	H,FF91
FF57	91		
FF58	FF		
FF59	0E	MVI	C,01
FF5A	01		
FF5B	23	INX	H
FF5C	23	INX	H
FF5D	7E	MOV	A,M
FF5E	B7	ORA	A

Continued on next page…

ADDRESS	DATA	DESCRIPTION	
FF5F	CA	JZ	FF68
FF60	68		
FF61	FF		
FF62	CD	CALL	FF7C
FF63	7C		
FF64	FF		
FF65	C3	JMP	FF5C
FF66	5C		
FF67	FF		
FF68	2B	DCX	H
FF69	2B	DCX	H
FF6A	7E	MOV	A,M
FF6B	B7	ORA	A
FF6C	CA	JZ	FF75
FF6D	75		
FF6E	FF		
FF6F	CD	CALL	FF7C
FF70	7C		
FF71	FF		
FF72	C3	JMP	FF69
FF73	69		
FF74	FF		
FF75	0D	DCR	C
FF76	CA	JZ	FF5B
FF77	5B		
FF78	FF		
FF79	C3	JMP	FF01
FF7A	01		
FF7B	FF		
FF7C	0C	INR	C
FF7D	0D	DCR	C
FF7E	C2	JNZ	FF85
FF7F	85		
FF80	FF		
FF81	47	MOV	B,A
FF82	3E	MVI	A,3F
FF83	3F		
FF84	90	SUB	B
FF85	D3	OUT	13
FF86	13		
FF87	20	RIM	

ADDRESS	DATA	DESCRIPTION	
FF88	E6	ANI	40
FF89	40		
FF8A	CA	JZ	FF87
FF8B	87		
FF8C	FF		
FF8D	3E	MVI	A,10
FF8E	10		
FF8F	30	SIM	
FF90	C9	RET	

From here down is sine wave data.

FF91	00
FF92	1F
FF93	21
FF94	23
FF95	25
FF96	27
FF97	29
FF98	2B
FF99	2D
FF9A	2E
FF9B	30
FF9C	32
FF9D	34
FF9E	35
FF9F	36
FFA0	38
FFA1	39
FFA2	3A
FFA3	3B
FFA4	3C
FFA5	3D
FFA6	3D
FFA7	3E
FFA8	3E
FFA9	3F
FFAA	3F
FFAB	3F
FFAC	00

Application 3: Interfacing a Temperature Sensor to the PRIMER

Purpose

To expose the student to rudimentary analog interface techniques.

Goals

1. Build and test a simple temperature sensing circuit.
2. Load a program that will make use of the temperature sensor's output.
3. Calibrate the sensor and software to provide a temperature reading in approximate engineering units.
4. Control a simple process with temperature.

Materials

Qty.	Description	DIGI-KEY part number
1	PRIMER trainer	
1	Fahrenheit thermometer	
1	hair dryer	
1	LM358 Dual Op-Amp.	LM358N
1	LM35 Prec. Celsius Temp Sensor	LM35DZ-ND
1	100Ω 1% metal film resistor	100.0XBK-ND
2	1 KΩ 1% metal film resistor	1.00KXBK-ND
4	100 KΩ 5% carbon film resistor	100KQBK-ND
1	100 KΩ Potentiometer	3292W-104-ND
1	8-pin soldertail dip socket	A9308
1"x2" piece of perfboard		

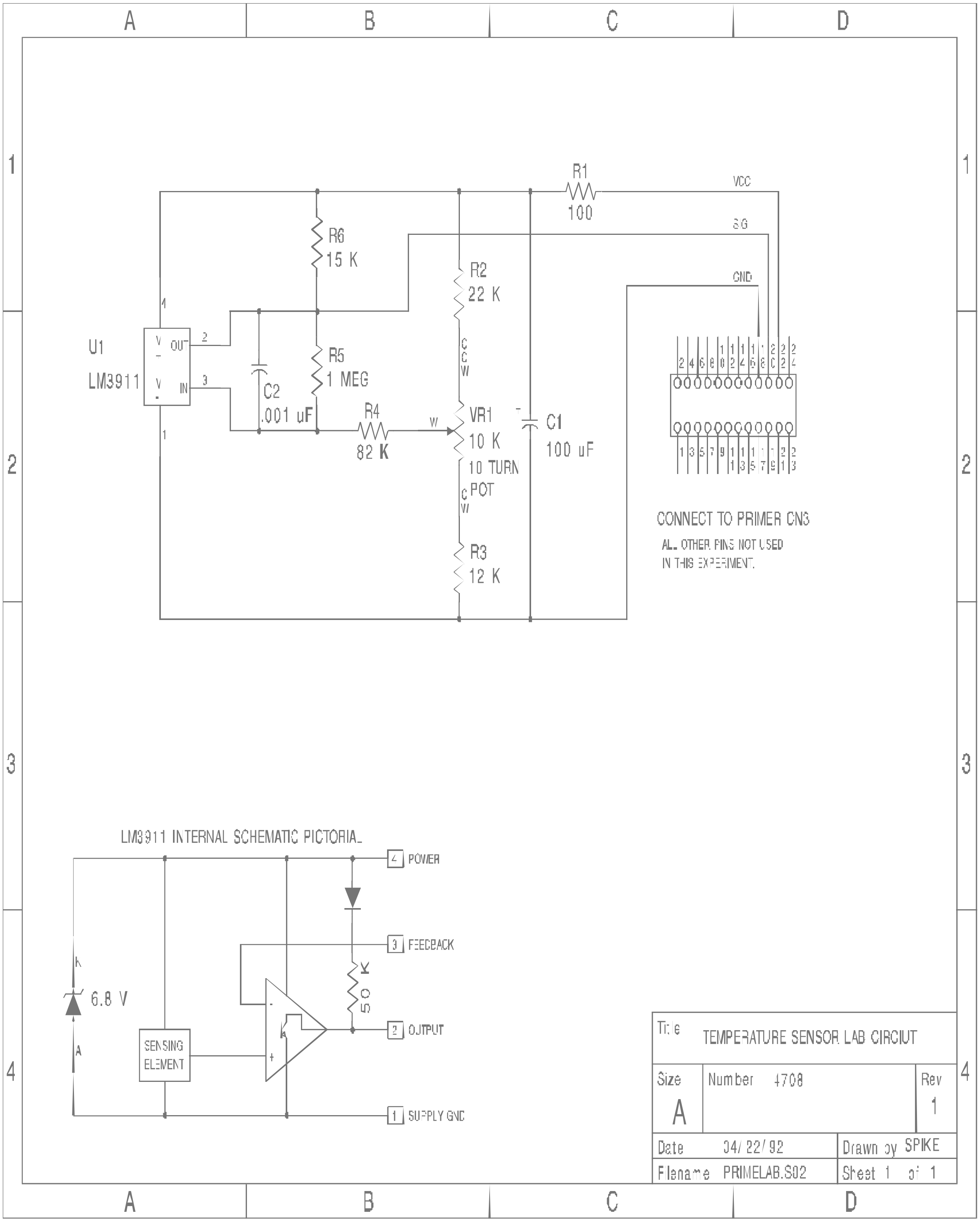
(A digital voltmeter may also prove helpful if available)

The electronic components listed above may be ordered from DIGI-KEY®, by telephone by dialing 1-800-344-4539. They may also be found at electronic supply stores and other mail order houses.

Circuit Description

The temperature sensing circuit used, in our application, is centered around the National Semiconductor LM35 series temperature sensors. The LM35N, with a range of (0 - 100 degrees Celsius), will be used in our application and produce an output voltage that is linearly proportional to the Celsius temperature. The LM35 senses temperature by amplifying the voltage differential at the base-emitter junctions of two identical transistors, that are operating at different currents, with the same temperature applied to them. As the junction temperature changes, the curve of base-emitter voltage vs. temperature will differ between the two transistors, because they are operating at different currents. This differential would normally be a problem in conventional circuitry, but is taken advantage of here. The differential voltage is amplified by the LM35, and presented to the output. The LM35, unlike other sensors, is calibrated in Celsius and provides 10 millivolts per degree Celsius. The advantage of this calibration is that we need not subtract a large constant voltage from the output to scale down Kelvin calibration. Each degree Kelvin is the same as one degree centigrade, but the scales start at different absolute temperatures. Zero degrees Kelvin is -273 degrees centigrade, therefore, 0 degrees centigrade is +273 degrees Kelvin. Additional Information may be obtained from National Semiconductors website at (http://www.national.com/pf/LM/LM35.html)

Although Kelvin and Celsius are equivalent (for this application) Fahrenheit degrees are entirely different. Both the scale shift, and the scale "gain" are different. Standard conversion formulas are used to convert centigrade to Fahrenheit and vice-versa. As nine Fahrenheit degrees pass for 5 Celsius degrees (5/9 plus



R1
100
VCC
SIG
GND
R6
15 K
R2
22 K
U1
LM3911
OUT
IN
C2
.001 uF
R5
1 MEG
R4
82 K
VR1
10 K
10 TURN
POT
C1
100 uF
R3
12 K
CONNECT TO PRIMER CN3
ALL OTHER PINS NOT USED
IN THIS EXPERIMENT.
LM3911 INTERNAL SCHEMATIC PICTORIAL
4 POWER
3 FEEDBACK
2 OUTPUT
1 SUPPLY GND
6.8 V
50 K
SENSING
ELEMENT
Title
TEMPERATURE SENSOR LAB CIRCIUT
Size
A
Number 4708
Rev
1
Date 04/22/92
Drawn by SPIKE
Filename PRIMELAB.S02
Sheet 1 of 1

the 32 Fahrenheit scale shift), each degree Fahrenheit will produce an eighteen (18) millivolt change per degree Fahrenheit. The program description describes how the analog reading is converted to Fahrenheit.

Referring to the schematic, the LM35 temperature sensor chip, U1, is powered by the 5 volt VCC supply of the PRIMER, which comes from the header connector plugged onto the analog port pins. As temperature rises, the LM35 output voltage (pin 2), rises. In our application, the PRIMER requires an inverse proportionality to the temperature rise. To achieve this inverse proportion to temperature rise , one half of U2, (LM358 Dual Op-Amp) is configured as a DC Summing Amplifier. The output of the LM35 is fed into the inverting pin (2), of the LM358. Pin 3 of the (LM358 Dual Op-Amp) has a voltage reference applied via VR1,R5,R6,R7. The output of the LM35 is subtracted from the voltage reference obtaining the inverse proportionality with temperature rise.

The PRIMER's A/D converter has 6 bits of resolution. This works out to 2^6 or 64 unique readings (or counts, as it is often termed in reference to A/D's) from 0 to 5V or 5V/64 = 0.078V per count which is 78mV per count. The circuit was designed to cause a change of slightly more than one count per millivolt change. To achieve this the second half of the LM358 is configured as a non-inverting DC amplifier. The output of the DC Summing Amplifier, via pin 1, is applied to the non-inverting pin, 5 . The gain is set via the feedback resistor, R1, and R2 and applied to the inverting pin 6. The resistor values for R1 and R2 have been chosen to provide a gain of 11 to the output via pin 7 and therefore will output 110 milivolts per degree Celsius.

Procedure

The temperature circuit should be built on perfboard, and connected to the PRIMER's analog port connector header. The circuit may be connected by wire-wrapping, soldering or by using a female connector. The circuit will draw power from the PRIMER, and feed its analog output to the PRIMER. Carefully check the wiring of the circuit, and be sure it is properly connected to the PRIMER.

HINT: Allow the circuit to thoroughly cool after soldering and handling. Residual heat that remains in the LM35 package, will deter attempts to adjust the setpoint correctly. If you set VR1, and the reading slowly drifts down, (lower temperature) it is probably due to this effect.

The assembly language code is listed below:

```
         ; This program shows the Fahrenheit temperature in the
         ; left four displays
leds     equ     11h        ; output port for digital output LEDs
adcin    equ     9          ; ADCIN service number
leddec   equ     13h        ; LEDDEC service number
mult     equ     7          ; MULT service number
div      equ     8          ; DIV service number
mos      equ     1000h      ; address of MOS services
adjst    equ     123        ; #of Fahrenheit degrees * 100 per
                            ; change in value returned from ADCIN

         org     0ff01h
loop:    mvi     c,adcin
         call    mos        ; get the digital value of analog input voltage
         mvi     h,0
         lda     mxanlg     ; maximum analog value (this may be different on
                            ; other PRIMERs, or with different temp sensors)
         sub     l          ; invert the analog conversion
         mov     l,a        ; HL = analog value
         lxi     d,adjst    ; load D with the adjustment factor
         mvi     c,mult
```

```
         call    mos         ; DE = HL * DE
         xchg                ; HL = DE
         lxi     d,100
         mvi     c,div
         call    mos         ; divide HL by 100
         lda     basetmp     ; get the base temperature
         add     l           ; now A is the actual temperature
         mov     e,a         ; E = temperature
         mov     a,e         ; A = temperature
         lhld    lotemp      ; L = low temp limit, H=high temp limit
         cmp     l           ; see if analog value is below L
         jnc     chkhi       ; check the high value if not
         mvi     a,0
         out     leds        ; turn on LEDs
chkhi:   mov     a,e         ; A = temperature
         cmp     h
         jc      noled       ; if A<H then don't turn off LEDs
         mvi     a,0FFh
         out     leds        ; H > = A so turn off LEDs
noled:   mvi     d,0         ; clear D register
         mvi     c,leddec
         call    mos         ; display the temp in DE
         jmp     loop        ; read it again

mxanlg:  ds      1           ; max analog value given by temp sensor
basetmp: ds      1           ; base temperature
lotemp:  ds      1           ; lower limit temperature
hitemp:  ds      1           ; upper limit temperature
         end
```

Load the following machine language program into memory:

ADDRESS	DATA	INSTRUCTION
FF01	0E	MVI C,09
FF02	09	
FF03	CD	CALL 1000
FF04	00	
FF05	10	
FF06	26	MVI H,00
FF07	00	
FF08	3A	LDA FF42
FF09	42	
FF0A	FF	
FF0B	95	SUB L
FF0C	6F	MOV L,A
FF0D	11	LXI D,007B
FF0E	7B	
FF0F	00	
FF10	0E	MVI C,07
FF11	07	
FF12	CD	CALL 1000
FF13	00	
FF14	10	
FF15	EB	XCHG
FF16	11	LXI D,0064
FF17	64	
FF18	00	

ADDRESS	DATA	INSTRUCTION
FF19	0E	MVI C,08
FF1A	08	
FF1B	CD	CALL 1000
FF1C	00	
FF1D	10	
FF1E	3A	LDA FF43
FF1F	43	
FF20	FF	
FF21	85	ADD L
FF22	5F	MOV E,A
FF23	7B	MOV A,E
FF24	2A	LHLD FF44
FF25	44	
FF26	FF	
FF27	BD	CMP L
FF28	D2	JNC FF2F
FF29	2F	
FF2A	FF	
FF2B	3E	MVI A,0
FF2C	00	
FF2D	D3	OUT 11
FF2E	11	

Continued on next page…

ADDRESS	DATA	INSTRUCTION
FF2F	7B	MOV A,E
FF30	BC	CMP H
FF31	DA	JC FF38
FF32	38	
FF33	FF	
FF34	3E	MVI A,FF
FF35	FF	
FF36	D3	OUT 11
FF37	11	
FF38	16	MVI D,00
FF39	00	
FF3A	0E	MVI C,13
FF3B	13	
FF3C	CD	CALL 1000
FF3D	00	
FF3E	10	
FF3F	C3	JMP FF01
FF40	01	
FF41	FF	
FF42	3F	(max analog val)
FF43	00	(base temp data)
FF44	5A	(lo temp limit)
FF45	64	(hi temp limit)

After loading in the program, you must calibrate the temperature sensor circuit and the program. Start the program running at FF01 and observe the left four numeric output LEDs. A decimal number should be displayed there. With a small screwdriver, turn the potentiometer (VR1) clockwise. If after 20 turns the output hasn't changed, turn VR1 counterclockwise for 20 turns (VR1 has mechanical stops that don't care if you turn them too many times). Adjust VR1 until the value on the display is as low as it can go. As soon as the value on the display stops decreasing, stop turning VR1. Subtract the value that is on the displays from 64 (decimal), stop the program then convert that value to hexadecimal and store it at FF42. Since the value returned by the A/D convertor decreases as the temperature increases, it is subtracted from the maximum value the A/D convertor can produce (normally 63 decimal) thereby inverting the value. The temperature sensor, though, does not produce the 5 volts required to give the maximum value, and for this reason the value at FF42 must be changed.

Now check the temperature of the sensor using a thermometer and convert this value to hex and store it at FF43. This is the base temperature. If you start the program at FF01 again, the base temperature (or within 1 or 2 degrees of it) will be shown on the displays. Heat up the sensor with the hair dryer and you will see that when the displayed temperature reaches 100 degrees the digital output LEDs turn off. Let the sensor cool down to below 90 degrees and they will turn on again. It is possible for the digital output connector (connected to the digital output LEDs) to control external devices such as fans or heaters, if you know how to build relay drivers that will turn such devices on and off (do not attempt this if you are not proficient in electronics). If a fan is connected to the output connector, the program can turn on the fan when the temperature reaches 100 degrees and turn it off when the temperature drops below 90 degrees. Likewise, if a heater is connected, the program can turn on the heater when the temperature drops below 90 and turn it off when the temperature reaches 100 degrees.

You may be wondering by now why the program is written in such a way as to turn the LEDs on at one temperature and turn them off at another. This is done to keep the output device from rapidly oscillating on and off. Rapid oscillation is fine when dealing with LEDs but it can be destructive to relays. This technique of using different turn on and turn off temperatures is commonly used in environment control systems. To see what would happen if there was one turn on and turn off temperature, store 5A at address FF45 and run the program. Heat up the sensor to 89 degrees and while watching the digital output LEDs, slowly heat the sensor to 90 degrees. You should see that as the temperature approaches 90 degrees the LEDs will start to oscillate rapidly for a moment (the LEDs may appear to dim) until the temperature is stable at 90 degrees.

Program Description

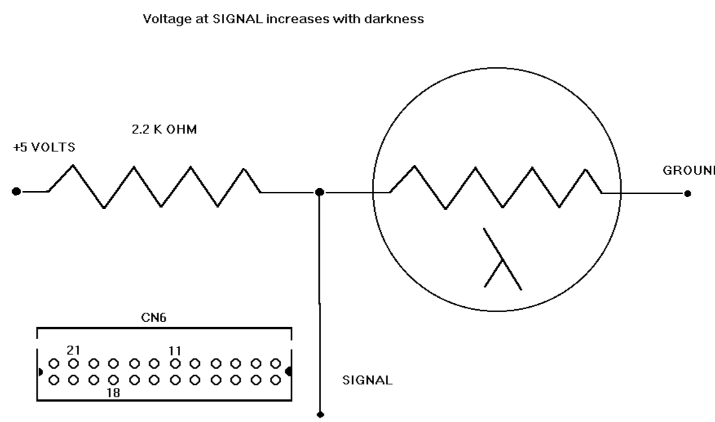
The program reads the analog to digital converter and then inverts the value that was returned from it so that as the temperature increases, the value will increase. This value is then scaled to provide an accurate Fahrenheit temperature. It was found through experimentation, that a change of 69 degrees from the base temperature causes the A/D converter value to change by 56 decimal. This means that for each change in A/D converter value there is a 69/56 or 1.23 degree change in the temperature. Since MOS only does integer math, a trick had to be used to perform floating point math. The inverted A/D converter value was multiplied by 123 and then the product was divided by 100 which effectively scaled the value by 1.23 and removed the tenths and hundredths digits. After the A/D converter value is converted to Fahrenheit, the base temperature is added to it to give the actual value. After this, it is compared to the low and high temperature values. If the temperature is below the low temperature value, zero is sent to the port for the digital output LEDs (which causes them to turn on), and if the temperature is at the high temperature limit, FF hex is sent the to the port (which turns the LEDs off). Finally the temperature is displayed on the left 4 displays and the program starts all over again.

Application 4: Interfacing a Photocell

This application shows how to interface a photocell to the PRIMER Trainer and gives an example program which demonstrates its capabilities.

Start out by getting the needed parts. These parts can be obtained from Radio Shack if desired. The circuit is so simple (see diagram) that you may build it without a perfboard.

Qty.	Description	DIGI-KEY part number
1	Cadmium Sulfide Photocell	276-118
1	2.2 KΩ 1/4 or 1/8 watt resistor	



The circuit is so simple (see diagram) that you may build it without a perfboard. You may connect it to CN 3 by wire-wrapping, soldering, or using a female connector (be sure to disconnect power from the PRIMER first). After building the circuit and connecting it to CN3, reconnect the power and see if the board powers up correctly. If it does not, disconnect power again and check the circuit. Once the board is powered up correctly, you will want to enter the self test mode by pressing "FUNC." then "1". After the RAM diagnostics are complete, the analog to digital conversion value will be displayed on the right two displays while a proportional tone is emitted from the speaker. In normal room lighting, the number displayed should be around 20 hex, and with the photocell darkened, the number should be close to 00.

If the circuit appears to be working correctly, press reset and proceed to the next page.

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION
8F01	AF	XRA A
8F02	32	STA 8FB2
8F03	B2	
8F04	8F	
8F05	26	MVI H,00
8F06	00	
8F07	11	LXI D,8FA1
8F08	A1	
8F09	8F	
8F0A	CD	CALL 8F8B
8F0B	8B	
8F0C	8F	
8F0D	3A	LDA 8FB2
8F0E	B2	
8F0F	8F	
8F10	16	MVI D,07
8F11	07	
8F12	47	MOV B,A
8F13	E6	ANI 0F
8F14	0F	
8F15	C6	ADI 30
8F16	30	
8F15	C6	ADI 30
8F16	30	
8F15	C6	ADI 30
8F16	30	
8F17	5F	MOV E,A
8F18	0E	MVI C,11
8F19	11	
8F1A	CD	CALL 1000
8F1B	00	
8F1C	10	
8F1D	78	MOV A,B
8F1E	0F	RRC
8F1F	0F	RRC
8F20	0F	RRC

Continued on next page…

ADDRESS	DATA	DESCRIPTION
8F21	0F	RRC
8F22	E6	ANI 0F
8F23	0F	
8F24	C6	ADI 30
8F25	30	
8F26	15	DCR D
8F27	5F	MOV E,A
8F28	0E	MVI C,11
8F29	11	
8F2A	CD	CALL 1000
8F2B	00	
8F2C	10	
8F2D	0E	MVI C,09
8F2E	09	
8F2F	1E	MVI E,00
8F30	00	
8F31	CD	CALL 1000
8F32	00	
8F33	10	
8F34	7D	MOV A,L
8F35	07	RLC
8F36	07	RLC
8F37	07	RLC
8F38	E6	ANI 07
8F39	07	
8F3A	3C	INR A
8F3B	4F	MOV C,A
8F3C	3E	MVI A,FF
8F3D	FF	
8F3E	B7	ORA A
8F3F	1F	RAR
8F40	0D	DCR C
8F41	C2	JNZ 8F3E
8F42	3E	
8F43	8F	
8F44	D3	OUT 40
8F45	40	
8F46	01	LXI B,8FB1
8F47	B1	
8F48	8F	
8F49	DB	IN 41
8F4A	41	
8F4B	E6	ANI 01
8F4C	01	
8F4D	C2	JNZ 8F5B
8F4E	5B	
8F4F	8F	
8F50	7D	MOV A,L
8F51	02	STAX B
8F52	11	LXI D,8FA8
8F53	A8	
8F54	8F	
8F55	CD	CALL 8F8B
8F56	8B	
8F57	8F	
8F58	C3	JMP 8F2D

ADDRESS	DATA	DESCRIPTION
8F59	2D	
8F5A	8F	
8F5B	0A	LDAX B
8F5C	C6	ADI F6
8F5D	F6	
8F5E	BD	CMP L
8F5F	DA	JC 8F64
8F60	64	
8F61	8F	
8F62	26	MVI H,01
8F63	01	
8F64	0A	LDAX B
8F65	BD	CMP L
8F66	D2	JNC 8F78
8F67	78	
8F68	8F	
8F69	24	INR H
8F6A	25	DCR H
8F6B	CA	JZ 8F78
8F6C	78	
8F6D	8F	
8F6E	21	LXI H,8FB2
8F6F	B2	
8F70	8F	
8F71	7E	MOV A,M
8F72	3C	INR A
8F73	B7	ORA A
8F74	27	DAA
8F75	77	MOV M,A
8F76	26	MVI H,00
8F77	00	
8F78	11	LXI D,0000
8F79	00	
8F7A	00	
8F7B	24	INR H
8F7C	25	DCR H
8F7D	C2	JNZ 8F83
8F7E	83	
8F7F	8F	
8F80	11	LXI D,0320
8F81	20	
8F82	03	
8F83	0E	MVI C,10
8F84	10	
8F85	CD	CALL 1000
8F86	00	
8F87	10	
8F88	C3	JMP 8F07
8F89	07	
8F8A	8F	
8F8B	E5	PUSH H
8F8C	C5	PUSH B
8F8D	EB	XCHG
8F8E	46	MOV B,M

Continued on next page…

ADDRESS	DATA	DESCRIPTION
8F8F	23	INX H
8F90	16	MVI D,00
8F91	00	
8F92	5E	MOV E,M
8F93	0E	MVI C,11
8F94	11	
8F95	CD	CALL 1000
8F96	00	
8F97	10	
8F98	14	INR D
8F99	23	INX H
8F9A	05	DCR B
8F9B	C2	JNZ 8F92
8F9C	92	
8F9C	8F	
8F9E	C1	POP B
8F9F	E1	POP H
8FA0	C9	RET

ADDRESS	DATA	DESCRIPTION
8FA1	06	DATA FOR "CELL->"
8FA2	43	
8FA3	45	
8FA4	4C	
8FA5	4C	
8FA6	2D	
8FA7	3E	
8FA8	08	DATA FOR "--LOAD--"
8FA9	2D	
8FAA	2D	
8FAB	4C	
8FAC	4F	
8FAD	41	
8FAE	44	
8FAF	2D	
8FB0	2D	
8FB1	64	SETPOINT
8FB2	00	COUNT

Application 5: Using the PRIMER to Regulate the Speed of a DC Motor

Purpose

To introduce the student to one method of regulating the speed of a small DC motor.

Goals

1. Study formulas, data, and waveforms relating to a DC motor.
2. Build an interface circuit that will allow the PRIMER to regulate the speed of a particular DC motor.
3. Build a motor holding fixture that will allow one motor to be mechanically coupled to another.
4. Load, run, and test a program that will allow the PRIMER via the interface circuit to:
   A. Regulate the speed of a particular DC motor.
   B. Accept desired speed input via the on-board DIP switches.
   C. Display motor speed and pulse width via the on-board 7-segment displays and LEDs respectively.

Equipment, Components, and Materials

Equipment (required):

Qty.	Description	Source	Part Number
1	PRIMER	EMAC	E600-00
1	Solderless Breadboard	Radio Shack	276-175
1	PRIMER Interface Cable	EMAC	E600-15

Components and Materials:

Interface Circuit

Qty.	Description	Source	Part Number
1	Transistor, 2N2222	Digi-Key	PN2222A-ND
1	Transistor, 2N2907	Digi-Key	PN2907A-ND
1	Resistor, 8.2 KΩ, ¼W, 5%, Carbon Film	Digi-Key	8.2KQ
1	Resistor, 1.8 KΩ, ¼W, 5%, Carbon Film	Digi-Key	1.8KQ
1	Resistor, 1 KΩ, ¼W, 5%, Carbon Film	Digi-Key	1.0KQ
1	Resistor, 390 Ω, ¼W, 5%, Carbon Film	Digi-Key	390Q
1	Diode, 1N4005	Digi-Key	1N4005GI
1	Capacitor, 2200 µF, 16V	Digi-Key	P1216

Motor Load Resistors

Qty.	Description	Source	Part Number
1	Resistor, 1.0 Ω, ½W, 5%, Carbon Film	Digi-Key	1.0H
1	Resistor, 3.3 Ω, ½W, 5%, Carbon Film	Digi-Key	3.3H
1	Resistor, 8.2 Ω, ½W, 5%, Carbon Film	Digi-Key	8.2H
1	Resistor, 33 Ω, ½W, 5%, Carbon Film	Digi-Key	33H

Motor Holding Fixture (optional)

Qty.	Description	Source	Part Number
1	Aluminum or Plexiglas Flat, 3.9" x 2.9" x 1/16-1/8"	-	-

2	Aluminum or Plexiglas Flat, 1.8" x 0.5" x 1/16-1/8"	-	-
8	Aluminum Spacers, Round Threaded, 4-40 x 0.75"	Digi-Key	J240
2	Perfboard, Glass epoxy, Pad per hole, 0.4" x 2.2"	-	-
2	Terminal Block, 2 position	Digi-Key	ED1631-ND
AR	Tennis Racquet Grip Wrap (Motor Mounting Pads) (or equivalent)	SOFTGRIP	STG-X
12	Pan Head Screws, 4-40 x 1/4"	Digi-Key	H142
4	Pan Head Screws, 4-40 x 1/2"	Digi-Key	H146
16	Lock Washers, #4	Digi-Key	H236
2	Motor with Gear (1.5 to 4.5VDC, 65mA @ 4.5VDC, 3 pole, permanent anisotropic magnet, 1.5 oz. in. stall torque)	Radio Shack	273-237

General

20" ea.	Wire, Stranded, 22 Ga., Red and Black	Radio Shack	278-1218
20"	Wire, Wire Wrap, 30 Ga.	Radio Shack	278-503

Introduction

In this lab, we would like to program the PRIMER to regulate the speed of a DC motor. The PRIMER will adjust motor speed by varying the armature voltage applied to the motor. This will be accomplished by varying the amount of time a fixed voltage is applied to the armature within a fixed time frame. This technique is called pulse width modulation (PWM). The time when voltage is applied to the motor will be referred to as "motor on time" or pulse width (PW). The time remaining in the fixed time frame would be "motor off time." The PRIMER will read the speed of the motor by using the on-board analog to digital (A/D) converter to measure the voltage (back EMF) generated by the motor during motor off time. This voltage is directly proportional to motor speed. By comparing motor speed to the desired speed, input via the on-board DIP switches, the PRIMER can correctly adjust motor on time to keep motor speed constant. Before we get to the interface circuit and PRIMER program needed to regulate motor speed, it might be helpful to look at some basic information relative to DC motors in general and to the motor we will be regulating in particular.

Motor Formulas

$$T = 7.04 K \Phi I_a$$

$$V_g = K \Phi N$$

$$I_a = V - \frac{V_g}{R_a}$$

$$N = \frac{V - I_a R_a}{K \Phi}$$

Where K = A constant for a particular motor.
Φ = Field Flux
I_a = Armature Current
R_a = Armature Resistance
V_g = Armature Voltage
N = Motor Speed
T = Motor Torque

These formulas show that there is a linear relationship between applied armature voltage *V* and motor speed *N* for a given load. Since back EMF, V_g, is directly related to motor speed there is also a linear relationship between *V* and V_g. The formulas also show that:

1. V_g will always be less than *V*.

2. I_a, and therefore torque are greatest at low motor speed and both decrease as motor speed is increased.
3. When an increased load is applied to a motor it must supply more torque. This in turn means that I_a must increase. If I_a increases motor speed will decrease. The only way to return the motor to its original speed is to increase the armature voltage *V*.

The motor we will use in this lab is a permanent magnet type. Permanent magnets provide the field flux *F*. Magnetic fields setup by current flowing in the armature windings cause the armature to rotate inside the magnetic fields set up by the permanent magnets. To maintain armature rotation, the direction of the armature magnetic fields must constantly change relative to the fixed direction of the magnetic fields of the permanent magnets. This function is provided by brushes riding on a commutator attached to the motor shaft that constantly changes the direction of current flow in the armature windings as the shaft rotates. In this mode of operation, we supply electrical energy to the motor in the form of armature current and the motor supplies mechanical energy in the form of shaft rotation. If we supply mechanical energy to the motor by rotating the shaft, the motor will supply electrical energy in the form of armature current. This armature current results from the armature windings cutting across the magnetic lines of force set up by the magnetic fields of the permanent magnets. This current as seen by an electrical load across the motor terminals would be alternating (AC) if not for the rectifying action of the commutator converting it to DC. In this mode of operation, the motor is acting as a generator and the resulting DC voltage measured across the motor terminals is called counter or back EMF. The amplitude of this voltage will depend on the electrical load attached to the motor terminals but for a given load, changes in this back EMF will be directly proportional to changes in the speed of the rotating armature.

Motor Waveforms

If we use a pulse generator to apply pulse width modulation to the circuit of Figure 1 and observe the resulting A/D signal on an oscilloscope, we would see the waveforms of Figure 2.

The three regions of interest in the waveforms are marked as A, B, and C. The period of the PWM signal is A + B + C. The motor on time is A and the motor off time is B + C. Region B in waveform B is a negative voltage generated by the collapsing magnetic field in the armature windings when armature current is cut off at the beginning of motor off time. If this voltage were not clamped by diode D1 to about -0.7V, it would be a very large negative voltage that could potentially damage the PRIMER A/D circuitry. Region C in Waveform B is the back EMF generated by the armature rotating in the magnetic field of the permanent magnets during motor off time. If the pulse width of the PWM signal is now increased we would see the waveforms of Figure 3. The motor speed will noticeably increase and the amplitude of the back EMF of Region C will be greater. Two things are of interest in observing the motor waveforms that will have a bearing on our motor controller program.

1. The back EMF voltage is not "straight line smooth" as we would like it to be, but rather is a varying signal riding on a DC level. The amplitude of the varying signal seems to increase with increasing motor speed (increased pulse width). We could filter this with our circuitry but it would be difficult since we would not want to filter the motor on time voltage. This would introduce an unwanted error in the back EMF. A better solution would be to digitally filter (average) the back EMF by totaling 16 back EMF samples and then dividing the total by 16.
2. The point in the PWM period where we will begin to sample the back EMF must be carefully chosen to avoid sampling the motor on time voltage or the negative voltage transition. A sample window must be set up that will start late enough to assure back EMF will be present during maximum PW, but not so late that the program can't finish executing the required amount of code before the start of the next PWM period.

Motor Speed vs. Pulse Width and the Motor as an Integrator

If we applied increasing pulse widths to the circuit of Figure 1, allowed the motor to accelerate up to speed and recorded the back EMF for each pulse width for various motor loads and plotted the results we would get a graph similar to the one in Figure 4.

You might be surprised to see that the relationship between applied pulse width and back EMF is not linear for many of the curves. The curves appear to go from logarithmic for an unloaded motor toward linear as motor load is increased. This seems to contradict the results we would predict if we use the motor formulas we looked at earlier.

The reason for this is that we are asking the motor to integrate the PWM signal into an armature voltage. We would expect that:

$$V = \frac{\text{PW}}{\text{PERIOD}} \times V_{Q2} \text{ collector during PW}$$

This is a linear relationship but this relationship only holds up if the acceleration (charge) and deceleration (discharge) times in the motor (integrator) are close to equal. The acceleration time (charge time) will be much shorter than deceleration time at no motor load because we are driving the armature up to speed and then allowing the armature to decelerate at its own pace. Deceleration is strictly load dependent. If there is no load on the motor the deceleration time is long, (relative to acceleration time), the integrator discharge time is long, and the curve is logarithmic. As the motor load increases (decreasing RL), the acceleration (charge) and deceleration (discharge) times become more nearly equal, the motor begins to act more like a true integrator, the armature voltage to PW relationship becomes linear, and the graph becomes linear.

To state the previous discussion another way, if the linear changes in PW were producing linear changes in armature voltage, the motor would be responding linearly. Look at the graph in Figure 5.

Notice the motor speed response vs. pulse width increase is linear, independent of motor load. These plots were produced by integrating the PWM signal externally and applying the resulting voltage via a power op-amp to the motor. Now the motor is behaving as the formulas predict because it is not required to integrate the PWM signal. Since our program will allow the PRIMER to measure motor speed with the A/D converter and then adjust the pulse width to the value necessary to obtain the desired speed, you might imagine that nonlinearity in the motor speed curves is unimportant.

Nonlinearity can make it more difficult for our program to control motor speed. Consider the curve for an unloaded motor (motors uncoupled) in Figure 4. Notice that a pulse width change of only 1 count, say from 6 to 7, can cause a speed change of more than 10. This means it will be difficult if not impossible for our program to make fine adjustments in motor speed since it can only make incremental (not fractional) changes to pulse width. Now look at the curve in Figure 4 for a motor load of 8.1 ohms. Now incremental changes in pulse width result in incremental changes in motor speed and as a result much finer adjustment of motor speed will be possible. So even though our program will do a fair job controlling motor speed when the motor is operating on one of the non linear curves, it will do a much better job controlling speed when the motor is operating on a more linear curve.

Motor Interface Circuit Description and Assembly

Capacitor C1 in Figure 6 provides energy during times of high armature current to prevent fluctuations of the 5V supply. Resistor R1 sets the base current of transistor Q1 when PWM is high. Transistor Q1 provides base current for transistor Q2 when PWM is high. Q2 base current is set by resistors R2 and R3. Resistor R2 prevents Q2 conduction as a result of Q1 leakage or low level transients. Q2 provides armature current for motor M1 when PWM is high. Diode D1 clamps the negative voltage spike generated by the collapsing magnetic field of the armature at Q2 turn off. Resistor R4 limits the current into the A/D converter during the negative voltage spike.

Two advantages of using pulse width modulation applied directly to the motor to control motor voltage are:

1. Relatively simple interface circuitry.
2. There is much less power dissipation because the controlling devices are switches (on or off).

The circuit in Figure 6 consists of easily available, inexpensive components. The circuit can be constructed on a solderless breadboard and wired to the PRIMER and motor using the PRIMER Interface Cable. The PWM and A/D connections can be wire-wrapped from the PRIMER CN3 connector to wire-wrap posts or stiff wires pushed into the breadboard. The motor leads should be short lengths (10 in. max.) of 22 ga. wire soldered to the motor tabs (no polarity) and then tinned on the other end so they will push into the breadboard holes.

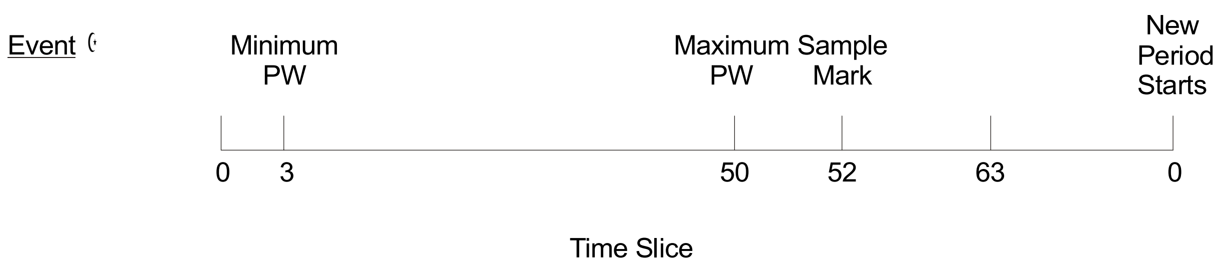
Motor Holding Fixture

A convenient way of loading one motor is to have it drive another motor which can in turn feed generated current through various load resistors to increase the load on the driving motor. If the motor you are using has a gear attached to the shaft, two motors can be coupled as illustrated in the motor fixture drawing. If your motor does not have a gear on the shaft, you can try coupling two motors with a short length of plastic tubing that will slip onto and hold tightly to the motor shafts. With this scheme the motors will be mounted in-line instead of offset in the motor fixture. Other motor loading schemes can be used such as using the motor to drive a propeller or placing a friction load against the motor shaft (holding your finger against the shaft at different degrees of pressure will do). You can choose your own method for mounting, coupling, and loading the motors but remember to construct fixtures from non-ferrous material because of the permanent magnets
in the motors.

Program Description

Refer to flowcharts 1 and 2 for a discussion of the motor controller program.

The program divides the PWM period into 64 time slices or t_slices. Each t_slice is 160µs long. The t_slices are numbered from 0-63. A variable called t_slice is incremented in an interrupt handler on every 7.5 interrupt. Continuous pulses 160µs apart from the timer chip initiate each 7.5 interrupt. This interrupt handler also manages the PWM output. If pulse width is less than time slice, PWM output (output port bit 0) is high, otherwise it's low. The scheduling of events is illustrated below:



The time between time slice 0 and sample mark is used to display speed and pulse width. These are displayed on the 7-segment LED display and LEDs 7-1 respectively. Notice there are upper and lower limits for pulse width. The time between maximum PW and sample mark is reserved to allow the negative voltage spike to pass when PW is maximum. The time between sample mark and end of period is used to sample the back EMF, average 16 samples, and calculate a new pulse width based on the current speed and the desired speed (set with the PRIMER DIP switches).

The program consists of two programs, a background program and a foreground program. The background program executes every time the microprocessor receives an interrupt pulse on the 7.5 interrupt pin. The timer chip is set by the initialization part of our program to provide a pulse to the 7.5 interrupt pin every 160µs. The background program has two functions.

1. To increment the time slice each time it executes. The only exception to this is when time slice reaches a maximum count of 63 at which time it is set back to zero.
2. To set the PWM signal (output port bit 0) high or low. If time slice is less than pulse width the output is high, otherwise it is low.

The foreground program monitors time slice and waits till it's 0. Then it displays motor speed on the leftmost four 7-segment LED digits and it displays pulse width in a bar graph fashion on LEDs 7-1 as follows:

Pulse Width	**LEDs On**	
0-7	(0% - 11%)	1
8-15	(12% - 23%)	1, 2
16-23	(24% - 36%)	1, 2, 3
24-31	(37% - 48%)	1, 2, 3, 4
32-39	(49% - 61%)	1, 2, 3, 4, 5
40-47	(62% - 73%)	1, 2, 3, 4, 5, 6
48-50	(74% - 78%)	1, 2, 3, 4, 5, 6, 7

The foreground program then waits for time slice to equal sample mark. Sample mark is set to accommodate the longest possible pulse width plus time for the negative voltage transition (after motor current cutoff) to expire. At sample mark the back EMF is sampled and added to a total of 16 such samples. If 16 samples have not yet been totaled the program repeats by going back and waiting for time slice to equal 0.

When 16 samples have been totaled, the total is divided by 16 to produce an average speed (it is this average speed that will later be displayed on the 7-segment display after time slice 0). The average speed is then subtracted from the speed set on the PRIMER DIP switches to produce an error term.

If the error is < -1, the pulse width is decremented.
If the error is > 1, the pulse width is incremented.
If the error is -1, 0, or 1, the *pulse width is unchanged.

The pulse width is then range checked. If the pulse width is less than minimum (3), it is set to minimum. If the pulse width is greater than maximum (50), it is set to maximum. Otherwise the pulse width is unchanged.

The entire process then repeats by going back and again waiting for time slice 0.

To test the motor speed program wire the circuit of Figure 6 and connect the PRIMER and drive motor M1 to the circuit as previously described. Couple the second motor M2 if available to the drive motor M1. Motor M2 if used should be unloaded (no RL across its terminals).

Set the PRIMER DIP switches for a speed of 20. Load the motor control program into the PRIMER and run the program. The motor will accelerate to speed and the PW and average speed will be displayed as previously described.

Load the drive motor by placing an 8.2Ω, ½W resistor across the terminals of motor M2 or by hand friction. The motor speed will decrease at first, as indicated by the 7-segment LED display. Then the PW will increase, as indicated by the 7 LEDs, to bring the motor speed back to 20.

Now remove the 8.2Ω load resistor from motor M2 or the friction source. The speed of the drive motor will increase suddenly and the PW will begin to decrease to bring the motor speed back to 20.

Use the curves of Figure 4 and load resistors for various speeds set in on the DIP switches to exercise the motor speed control program. Notice from the curves of Figure 4 that there are limits on the maximum speed attainable for various motor loads. If you try to request a motor speed greater than the motor can provide for a given load, the program will simply increase the pulse width to maximum to get the maximum speed possible.

The assembly language code is listed below:

```
; ----------------------------------------------------------------
; This program regulates the speed of a DC motor by...
; [1] Averaging 16 samples of back EMF during motor off time.
; [2] Generating an error term (DIP switch - average EMF).
; [3] Using the error term to adjust the pulse width.
; [4] Using the resulting pulse width to pulse width modulate
;             (PWM) the motor.
;
; WARNING:    Use a 9V supply with a current limit of 1000 mA or
;             more with this lab. The standard 500mA supply will
;             be damaged if it is used with this lab.
;
MOS:          EQU    1000H          ; MOS SERVICES ADDRESS.
PWM_PORT:     EQU    11H            ; DIGITAL OUTPUT PORT.
DIP_SW:       EQU    12H            ; DIP SWITCH PORT.
SERV09:       EQU    09H            ; MOS SERVICE.ADCIN => L.
SERV13:       EQU    13H            ; MOS SERVICE.DE => 7-SEG DISPLAY.
PW_MIN:       EQU    03H            ; MINIMUM PW. T=160uS X PW_MIN
PW_MAX:       EQU    32H            ; MAXIMUM PW. T=160uS X PW_MAX
MAX_SLICE:    EQU    3FH            ; MAXIMUM NUMBER OF TIME SLICES.
                                    ; SETS PWM PERIOD.
                                    ; T=160uS X MAX_SLICE.
SMARK:        EQU    34H            ; TIME SLICE WHERE BACK EMF
                                    ; SAMPLE WILL BE TAKEN.
VEC7HLF:      EQU    0FFE9H         ; 7.5 INTERRUPT VECTOR.
SCALELO:      EQU    35H            ; MODE/SCALER FOR TIMER,
SCALEHI:      EQU    11000000B      ; CONTINUOUS PULSES EVERY 160uS.
TIMERLO:      EQU    14H            ; TIMER PORT.
TIMERHI:      EQU    15H            ; TIMER PORT.
TIMCMD:       EQU    0CDH           ; TIMER CONTROL COMMAND.
CMDREG:       EQU    10H            ; TIMER CONTROL PORT.
INTMASK:      EQU    1AH            ; INTERRUPT MASK.

              ORG    0FF01H

              DI
              LXI    H,SLICER       ; POINT 7.5 INTERRUPT
              SHLD   VEC7HLF        ; VECTOR TO SLICER.
              MVI    A,SCALELO      ; SET UP TIMER FOR
              OUT    TIMERLO        ; CONTINUOUS PULSES
              MVI    A,SCALEHI      ; AT DESIRED INTERRUPT
              OUT    TIMERHI        ; RATE.
              MVI    A,TIMCMD
              OUT    CMDREG
              MVI    A,INTMASK
              SIM
              EI

PWM_MOTOR:
              LXI    H,0000H        ; REG H = TOTAL
              MVI    B,10H          ; REG B = SAMPLE COUNT.
```

```
CHKZERO:
                LDA     T_SLICE         ; TIME SLICE = 0 ?
                CPI     00H
                JNZ     CHKZERO         ; NO.GO CHECK SMARK.
                MVI     D,00H           ; DISPLAY SPEED.
                MOV     E,C             ; C = SPEED.
                PUSH    B
                MVI     C,SERV13
                CALL    MOS
                POP     B
                LDA     PULSE_WIDTH
                MOV     D,A             ; DISPLAY PW.
                MVI     E,0FFH          ; E = MASK.
                ORA     E               ; CLEAR CARRY.
ROT_MASK:
                RAL                     ; ROTATE 0 TO MASK.
                MOV     E,A             ; SAVE MASK.
                MOV     A,D             ; GET PW.
                SUI     08H             ; PW = PW - 8.
                MOV     D,A             ; SAVE RESULT TO D.
                MOV     A,E             ; GET MASK.
                JNC     ROT_MASK        ; PW STILL POS. ?
                DI                      ; DISABLE INTERRUPT.
                LDA     IMAGE           ; GET IMAGE.
                RAR                     ; SAVE BIT 0.
                MOV     A,E             ; GET MASK.
                RAL                     ; 7 BITS MASK + BIT 0.
                STA     IMAGE           ; TO IMAGE.
                EI                      ; ENABLE INTERRUPT.
CHK_SMARK:
                LDA     T_SLICE
                CPI     SMARK           ; TIME SLICE = SMARK ?
                JNZ     CHK_SMARK       ; NO.WAIT TILL IT IS.
                XCHG                    ; DE = TOTAL.
                PUSH    B               ; SAMPLE BACK EMF.
                MVI     C,SERV09
                CALL    MOS
                POP     B
                MVI     H,00H           ; HL = SAMPLE.
                DAD     D               ; HL = TOTAL + SAMPLE.
                DCR     B               ; DEC. SAMPLE COUNT.
                JNZ     CHKZERO         ; IF NOT 0, CHK 0 T_SLICE.

DIV_MORE:       DAD     H               ; HL*16/256=HL/16, SO...
                DAD     H               ; ...4 DAD H's MAKES HL*16...
                DAD     H               ; ..AFTER THIS H=HL/256 (THINK ABOUT IT)
                DAD     H               ; SPEED=TOTAL / MAX SAMP (16).
                MOV     C,H             ; STORE SPEED.
                IN      DIP_SW          ; GET DESIRED SPEED.
                ANI     00111111B       ; DES.SPEED 6 BITS MAX.
                SUB     H               ; SWITCH-SPEED=ERROR.
                LXI     H,PULSE_WIDTH
                JM      DECPW_CHK       ; ERROR = -. DEC PW ?
                CPI     2               ; ERROR < 2 ?
                JC      PW_RANGE        ; YES. NO PW CHANGE.
                INR     M               ; NO. INC PW.
                JMP     PW_RANGE        ; RANGE CHECK PW.
```

```
DECPW_CHK:
              CPI     0FFH           ; ERROR = -1.
              JZ      PW_RANGE       ; YES. RANGE CHECK PW.
              DCR     M              ; NO. DEC PW.
PW_RANGE:
              MVI     A,PW_MIN       ; PW < MIN ?
              CMP     M
              JC      MAX_CHK        ; NO. CHECK MAX.
              MOV     M,A            ; YES. PW = MIN.
MAX_CHK:
              MVI     A,PW_MAX       ; PW > MAX ?
              CMP     M
              JNC     PWM_MOTOR      ; NO. PW OK.
              MOV     M,A            ; YES. PW = MAX.
              JMP     PWM_MOTOR      ; START AGAIN.

; ----------------------------------------------------------------
; .......................SLICER...................................
; SLICER IS AN INTERRUPT HANDLER FOR THE 7.5 INTERRUPT.
; SLICER CONTROLS A TIME MARKER (T_SLICE) BY ADJUSTING IT FROM
; 0 TO MAX_SLICE IN EQUAL TIME INCREMENTS ON EACH 7.5 INTERRRUPT.
; SLICER ALSO CONTROLS THE WIDTH OF THE PULSE USED TO DRIVE THE
; MOTOR BY COMPARING THE VALUE OF PULSE_WIDTH TO THAT OF T_SLICE
; TO DETERMINE IF THE PULSE SHOULD BE HIGH OR LOW.
; PULSE HIGH => T_SLICE < PULSE_WIDTH.
; PULSE LOW => T_SLICE >=PULSE_WIDTH.
; ----------------------------------------------------------------

SLICER:
              PUSH    PSW            ; SAVE REGISTERS.
              PUSH    H
              LXI     H,T_SLICE      ; H POINTS TO T_SLICE.
              INR     M              ; INCREMENT T_SLICE
              MVI     A,MAX_SLICE
              CMP     M              ; T_SLICE = MAX_SLICE ?
              JNZ     PWM            ; NO. T_SLICE OK.
              MVI     M,00H          ; YES. T_SLICE = 0.
PWM:
              MOV     A,M            ; A = T_SLICE.
              LXI     H,PULSE_WIDTH  ; M = PULSE WIDTH.
              CMP     M              ; T_SLICE < PULSE WIDTH ?
              LXI     H,IMAGE        ; M = IMAGE.
              MOV     A,M            ; GET IMAGE.
              RAR                    ; CY => BIT 7.
              RLC                    ; BIT 7 => BIT 0.
              MOV     M,A            ; STORE IMAGE.
              OUT     PWM_PORT       ; OUTPUT IMAGE.
              POP     H              ; RECOVER REGISTERS.
              POP     PSW
              EI
              RET                    ; RETURN

T_SLICE:      DB      00H
PULSE_WIDTH:  DB      PW_MIN
IMAGE:        DS      01H
              END
; -----------------------------------------------------------
```

Load the following machine language program into memory:

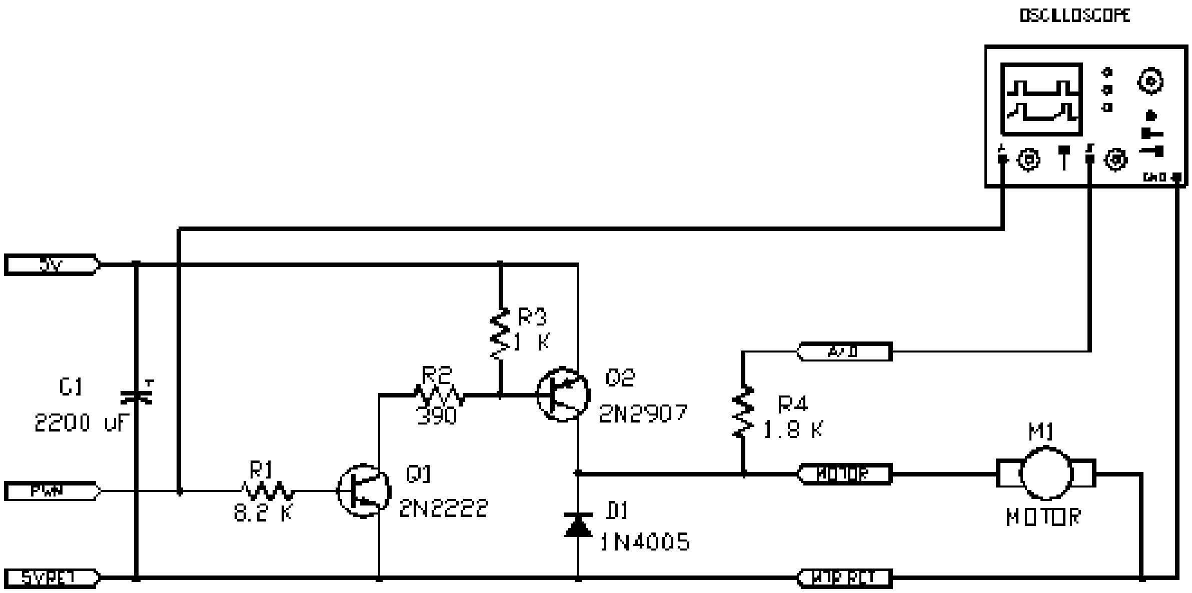
ADDRESS	DATA	DESCRIPTION	
FF01	F3	DI	
FF02	21	LXI	H,FF92
FF03	92		
FF04	FF		
FF05	22	SHLD	FFE9
FF06	E9		
FF07	FF		
FF08	3E	MVI	A,35
FF09	35		
FF0A	D3	OUT	14
FF0B	14		
FF0C	3E	MVI	A,C0
FF0D	C0		
FF0E	D3	OUT	15
FF0F	15		
FF10	3E	MVI	A,CD
FF11	CD		
FF12	D3	OUT	10
FF13	10		
FF14	3E	MVI	A,1A
FF15	1A		
FF16	30	SIM	
FF17	FB	EI	
FF18	21	LXI	H,0000
FF19	00		
FF1A	00		
FF1B	06	MVI	B,10
FF1C	10		
FF1D	3A	LDA	FFB2
FF1E	B2		
FF1F	FF		
FF20	FE	CPI	00
FF21	00		
FF22	C2	JNZ	FF1D
FF23	1D		
FF24	FF		
FF25	16	MVI	D,00
FF26	00		
FF27	59	MOV	E,C
FF28	C5	PUSH	B
FF29	0E	MVI	C,13
FF2A	13		
FF2B	CD	CALL	1000
FF2C	00		
FF2D	10		
FF2E	C1	POP	B
FF2F	3A	LDA	FFB3
FF30	B3		
FF31	FF		
FF32	57	MOV	D,A
FF33	1E	MVI	E,FF
FF34	FF		

ADDRESS	DATA	DESCRIPTION	
FF35	B3	ORA	E
FF36	17	RAL	
FF37	5F	MOV	E,A
FF38	7A	MOV	A,D
FF39	D6	SUI	08
FF3A	08		
FF3B	57	MOV	D,A
FF3C	7B	MOV	A,E
FF3D	D2	JNC	FF36
FF3E	36		
FF3F	FF		
FF40	F3	DI	
FF41	3A	LDA	FFB4
FF42	B4		
FF43	FF		
FF44	1F	RAR	
FF45	7B	MOV	A,E
FF46	17	RAL	
FF47	32	STA	FFB4
FF48	B4		
FF49	FF		
FF4A	FB	EI	
FF4B	3A	LDA	FFB2
FF4C	B2		
FF4D	FF		
FF4E	FE	CPI	34
FF4F	34		
FF50	C2	JNZ	FF4B
FF51	4B		
FF52	FF		
FF53	EB	XCHG	
FF54	C5	PUSH	B
FF55	0E	MVI	C,09
FF56	09		
FF57	CD	CALL	1000
FF58	00		
FF59	10		
FF5A	C1	POP	B
FF5B	26	MVI	H,00
FF5C	00		
FF5D	19	DAD	D
FF5E	05	DCR	B
FF5F	C2	JNZ	FF1D
FF60	1D		
FF61	FF		
FF62	29	DAD	H
FF63	29	DAD	H
FF64	29	DAD	H
FF65	29	DAD	H
FF66	4C	MOV	C,H

Continued on next page…

ADDRESS	DATA	DESCRIPTION	
FF67	DB	IN	12
FF68	12		
FF69	E6	ANI	3F
FF6A	3F		
FF6B	94	SUB	H
FF6C	21	LXI	H,FFB3
FF6D	B3		
FF6E	FF		
FF6F	FA	JM	FF7B
FF70	7B		
FF71	FF		
FF72	FE	CPI	02
FF73	02		
FF74	DA	JC	FF81
FF75	81		
FF76	FF		
FF77	34	INR	M
FF78	C3	JMP	FF81
FF79	81		
FF7A	FF		
FF7B	FE	CPI	FF
FF7C	FF		
FF7D	CA	JZ	FF81
FF7E	81		
FF7F	FF		
FF80	35	DCR	M
FF81	3E	MVI	A,03
FF82	03		
FF83	BE	CMP	M
FF84	DA	JC	FF88
FF85	88		
FF86	FF		
FF87	77	MOV	M,A
FF88	3E	MVI	A,32
FF89	32		
FF8A	BE	CMP	M
FF8B	D2	JNC	FF18
FF8C	18		
FF8D	FF		
FF8E	77	MOV	M,A

ADDRESS	DATA	DESCRIPTION	
FF8F	C3	JMP	FF18
FF90	18		
FF91	FF		
FF92	F5	PUSH	PSW
FF93	E5	PUSH	H
FF94	21	LXI	H,FFB2
FF95	B2		
FF96	FF		
FF97	34	INR	M
FF98	3E	MVI	A,3F
FF99	3F		
FF9A	BE	CMP	M
FF9B	C2	JNZ	FFA0
FF9C	A0		
FF9D	FF		
FF9E	36	MVI	M,00
FF9F	00		
FFA0	7E	MOV	A,M
FFA1	21	LXI	H,FFB3
FFA2	B3		
FFA3	FF		
FFA4	BE	CMP	M
FFA5	21	LXI	H,FFB4
FFA6	B4		
FFA7	FF		
FFA8	7E	MOV	A,M
FFA9	1F	RAR	
FFAA	07	RLC	
FFAB	77	MOV	M,A
FFAC	D3	OUT	11
FFAD	11		
FFAE	E1	POP	H
FFAF	F1	POP	PSW
FFB0	FB	EI	
FFB1	C9	RET	
FFB2	00	(time slice)	
FFB3	03	(pulse width)	
FFB4	xx	(output port, undefined leave blank)	



Schematic 1

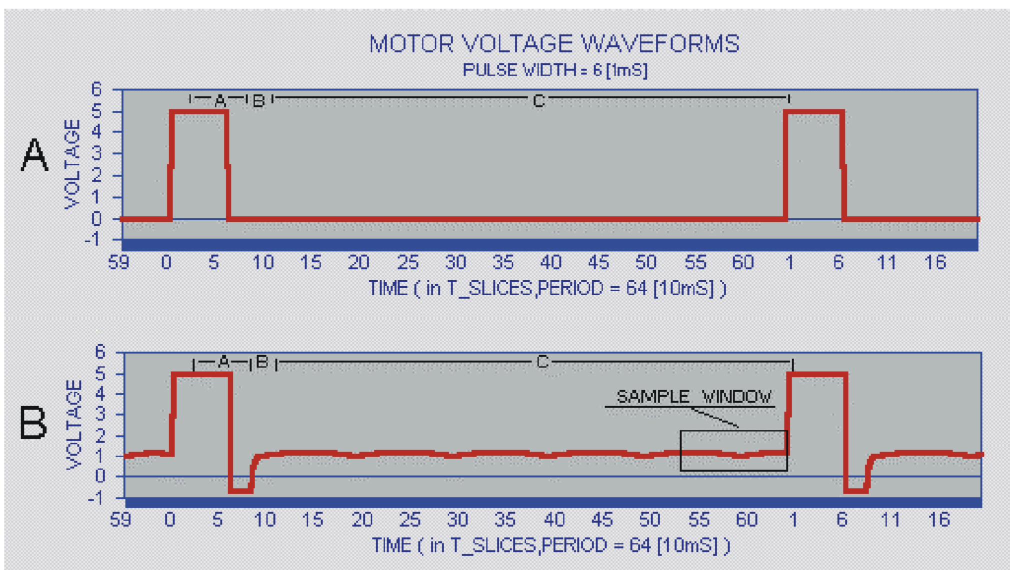


Figure 2

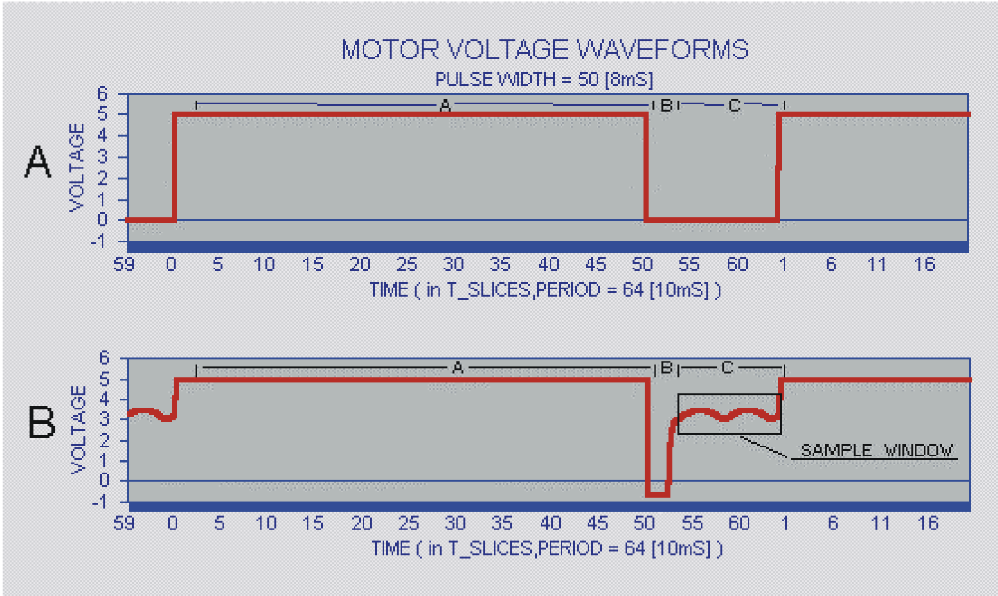


Figure 3

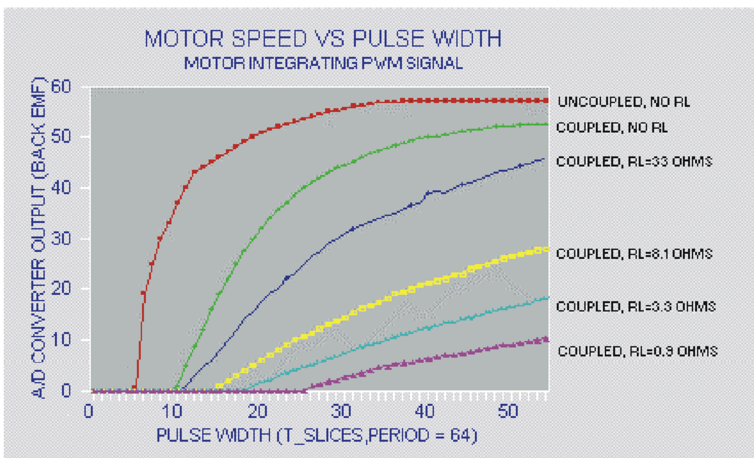


Figure 4

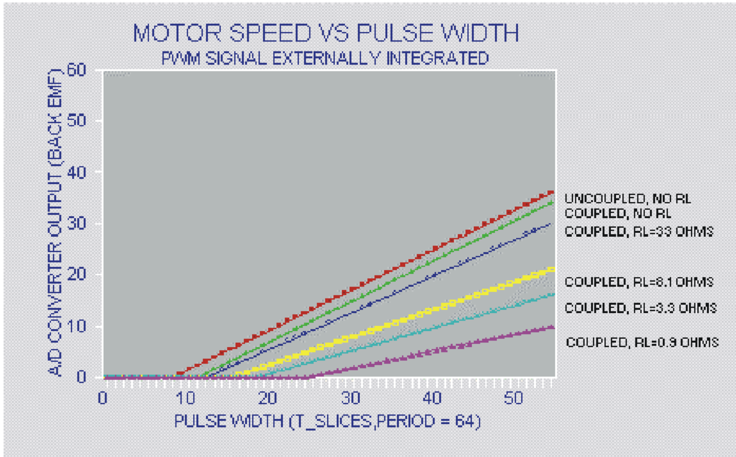
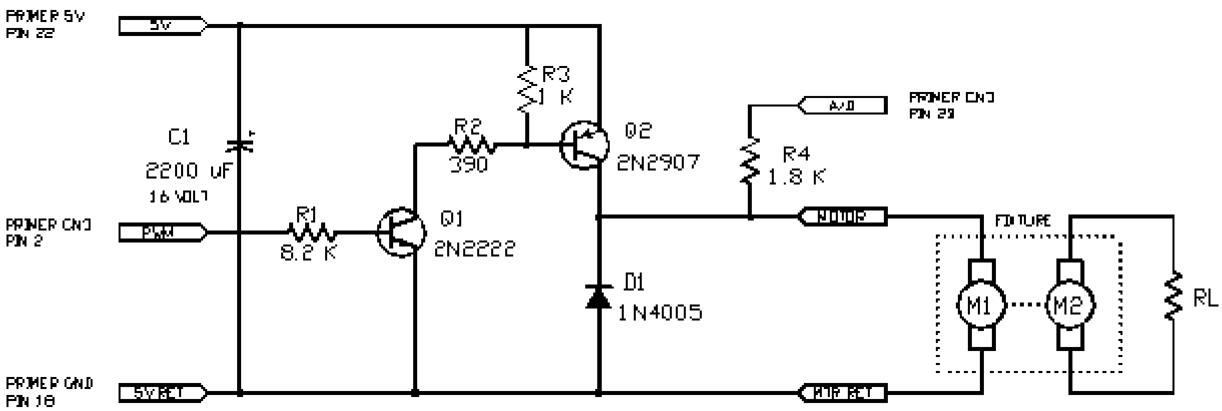


Figure 5



Schematic 2

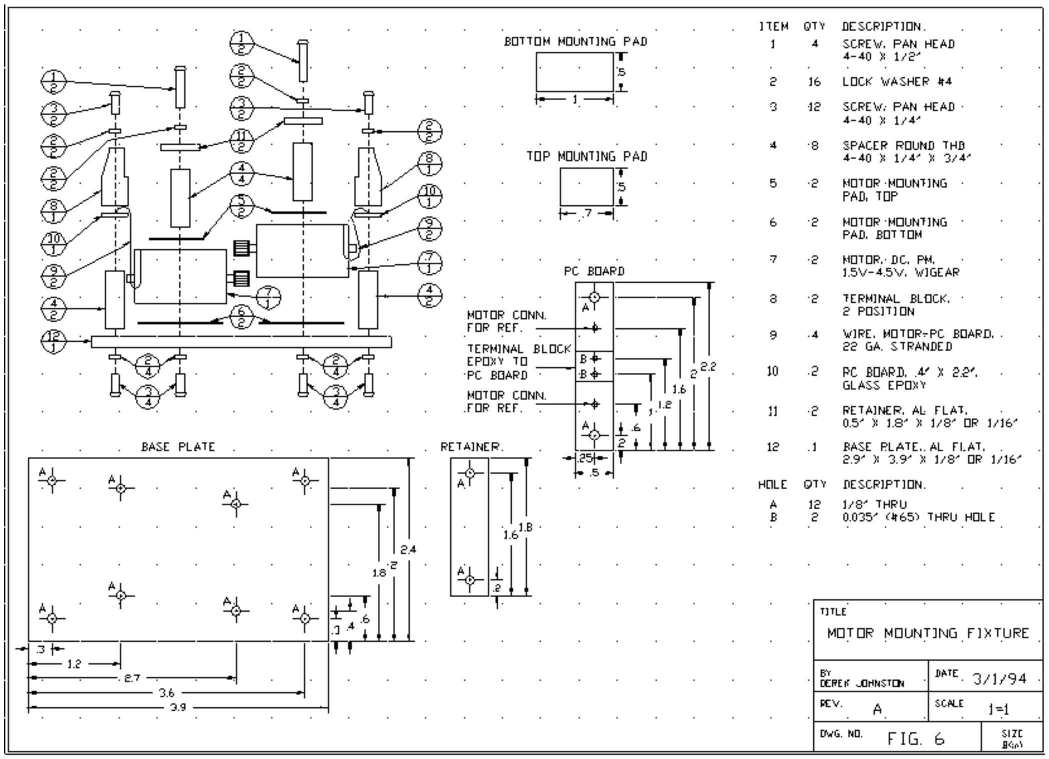
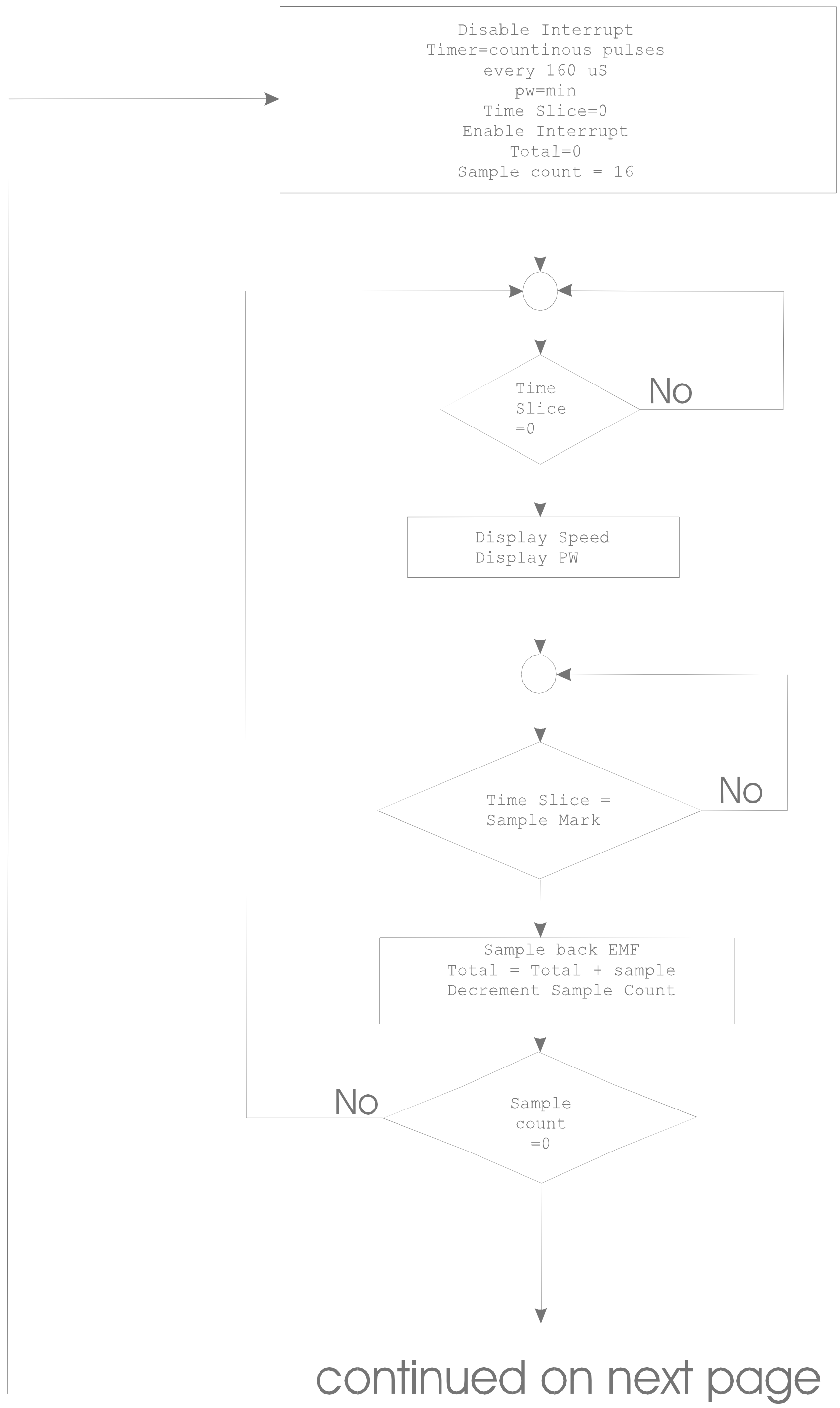

BOTTOM MOUNTING PAD
TOP MOUNTING PAD
PC BOARD
MOTOR CONN. FOR REF.
TERMINAL BLOCK EPOXY TO PC BOARD
MOTOR CONN. FOR REF.
BASE PLATE
RETAINER
ITEM QTY DESCRIPTION.
1 4 SCREW, PAN HEAD 4-40 X 1/2"
2 16 LOCK WASHER #4
3 12 SCREW, PAN HEAD 4-40 X 1/4"
4 8 SPACER ROUND THD 4-40 X 1/4" X 3/4"
5 2 MOTOR MOUNTING PAD, TOP
6 2 MOTOR MOUNTING PAD, BOTTOM
7 2 MOTOR, DC, PM, 1.5V-4.5V, W/GEAR
8 2 TERMINAL BLOCK, 2 POSITION
9 4 WIRE, MOTOR-PC BOARD, 22 GA. STRANDED
10 2 PC BOARD, .4" X 2.2", GLASS EPOXY
11 2 RETAINER, AL FLAT, 0.5" X 1.8" X 1/8" OR 1/16"
12 1 BASE PLATE, AL FLAT, 2.9" X 3.9" X 1/8" OR 1/16"
HOLE QTY DESCRIPTION.
A 12 1/8" THRU
B 2 0.035" (#65) THRU HOLE
TITLE
MOTOR MOUNTING FIXTURE
BY DEREK JOHNSTON
DATE 3/1/94
REV. A
SCALE 1=1
DWG. NO. FIG. 6

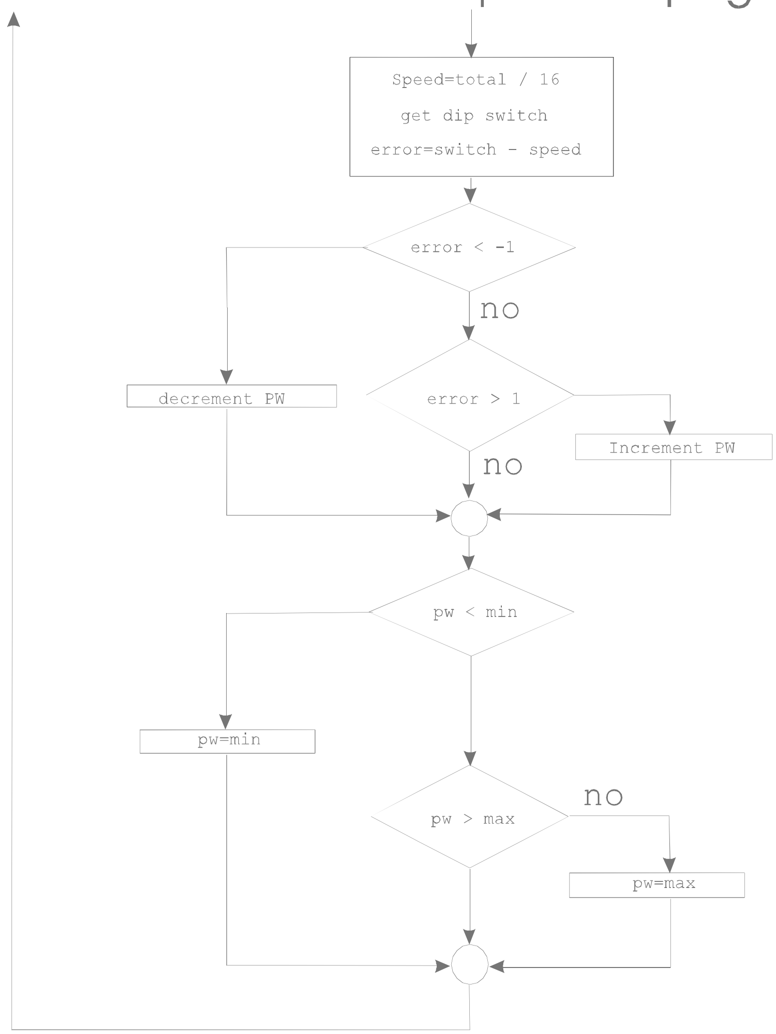
Figure 6



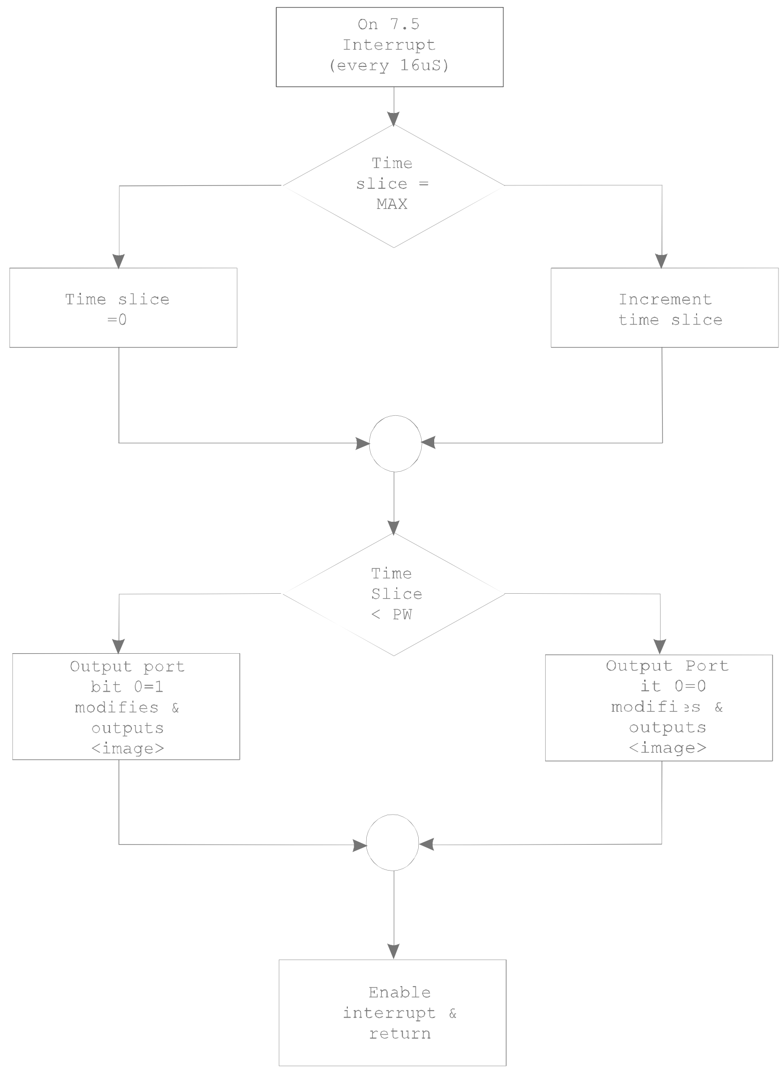
continued on next page

Flowchart 1
Foreground Motor Speed Control Program

Continued from previous page



Flowchart 1
Foreground Motor Speed Control Program (Continued)



Flowchart 2
Background Motor Speed Control Program

Application 6: External Multiplexed Display and Keypad Decoder

Purpose

To demonstrate and emulate the functions of a keypad and two digit LED display controller.

Goals

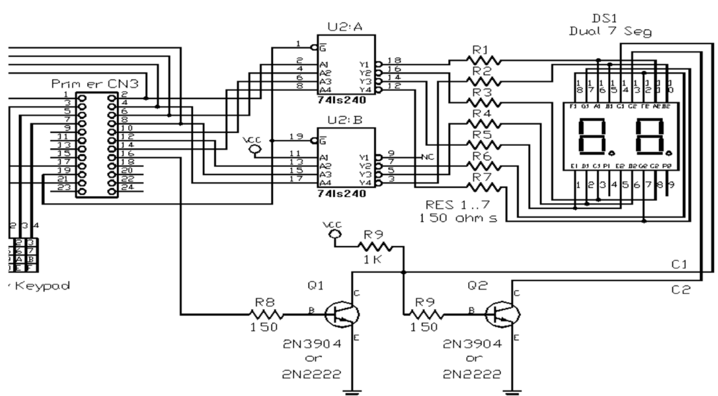
1. Build and test a keypad and numeric LED display interface.
2. Load a program that will demonstrate the numeric LED display interface.
3. Modify the program and load additional code which will demonstrate the keypad decoder.

Materials

Qty.	Description	Digi-Key Part Number
2	2N3904 or 2N2222	2N3904-ND or 2N2222-ND
1	74ls240	DM74LS240N-ND
1	4x4 matrix keypad	GH5004-ND
1	2-digit LED display	P355-ND
9	150 Ω 5% 1/4 watt resistor	
1	1 KΩ 5% 1/4 watt resistor	

This application will be demonstrated in two phases: with the display only, and then with the keypad and display.

Display Controller Circuit Description



To drive an external 7 segment display using the trainer, the 8 output lines (numbered 0 to 7) would be the obvious choice. This would provide control for each of the 7 elements leaving one output line free. What if we want to drive two digits?. We need 7 more outputs which we don't have. The answer to this problem is to use a multiplexed scheme of driving the digits.

We can drive the anodes of each of the elements of the pair of 7 segment displays with the same outputs (one output per matching pair of segments) and use the 8th (bit 7) to select which display will turn on by driving the cathode of the desired digit to ground. This will allow us to display data on the left digit and turn the right one off, and vice-versa. If this is done rapidly enough it will appear as if both digits are showing simultaneously, due to "persistence of vision" in the human eye.

To lessen the load on the output port, the outputs drive a 74LS240 tri-state inverting buffer and the outputs of this go to the anodes of both digits of the display. The buffer's two enable lines are tied to the Primer's digital to analog (D/A) output and they tri-state the outputs when the D/A is output is 5V. This turns off the display which will be necessary when including the keypad in the circuit. When the D/A output is 0V the buffer is enabled and the outputs go to the opposite logic level as their respective inputs.

If the buffer is enabled, bit 7 selects which display to turn on. If bit 7 is high, the voltage applied to the base of Q1 will bring the cathode for the left display to ground, causing it to turn on. When this happens, the base of Q2 is pulled to ground causing it to turn off, which turns off the display on the right. When bit 7 is low, this turns off Q1 which allows the base voltage of Q2 to rise and turn on the display on the right.

The assembly language code is listed below:

```
;
; External Multiplexed Display and Keypad Decoder program.
;
OPORT       EQU     11H         ; OUTPUT PORT
IPORT       EQU     12H         ; INPUT PORT
MOS         EQU     1000H       ; MOS CALL ADDRESS
DACSRV      EQU     0EH         ; D/A SERVICE

            ORG     0FF01H

LOOP:       IN      IPORT       ; READ DIP SWITCHES
            MOV     B,A
            CALL    HEXOUT      ; DISPLAY B
            JMP     LOOP
;
; Display the hex value of B on the LEDs. This routine must be
; called repeatedly in order for the data to be shown continuously,
; since it works on the principle of persistence of vision. The right
; digit is turned on and off first, then the left digit is turned on and off.
;
HEXOUT:     MOV     A,B         ; GET VALUE
            ANI     0FH         ; MASK OFF UPPER NIBBLE
            CALL    BIN7SG      ; CHANGE TO 7 SEG VALUE
            OUT     OPORT       ; SEND TO PORT
            CALL    FLSHDG      ; TURN ON DISPLAY MOMENTARILY

            MOV     A,B         ; GET ORIGINAL VALUE
            ANI     0F0H        ; NOW MASK OFF LOWER NIBBLE
            RRC
            RRC
            RRC
            RRC
            CALL    BIN7SG      ; CHANGE TO 7 SEG VALUE
            ORI     80H         ; SET BIT 7 SO LEFT DIGIT IS DISPLAYED
            OUT     OPORT       ; SEND TO PORT
            CALL    FLSHDG      ; TURN ON DISPLAY MOMENTARILY
            RET
;
; Change the binary number in A to its 7 seg. output pattern.
;
BIN7SG:     PUSH    H
            PUSH    D
            LXI     D,TAB7SG    ; POINT TO START OF TABLE
            MVI     H,0
            MOV     L,A         ; HL = OFFSET INTO TABLE
            DAD     D           ; ADD TABLE ADDR TO OFFSET
            MOV     A,M         ; GET OUTPUT PATTERN
            POP     D
            POP     H
            RET
```

```
;
; TRANSLATE TABLE FOR LED OUTPUT
;
TAB7SG:     DB      40H,79H,24H,30H
            DB      19H,12H,02H,78H
            DB      00H,18H,08H,03H
            DB      46H,21H,06H,0EH
;
; This flashes on and off the digit selected by bit 7 sent to OPORT.
;
FLSHDG:     PUSH    D
            PUSH    PSW
            CALL    LEDON       ; ENABLE LEDS
            LXI     D,0FFH
DELAY1:     DCX     D
            MOV     A,D
            ORA     E
            JNZ     DELAY1
            CALL    LEDOFF      ; DISABLE LEDS
            POP     PSW
            POP     D
            RET
;
; LEDON, LEDOFF, TURN ON/OFF THE LEDS THROUGH THE D/A OUTPUT
; 5V OUT TRI-STATES THE OUTPUTS OF THE 74LS240
; 0V OUT ENABLES THE OUTPUTS OF THE 74LS240
;
LEDON:      MVI     E,0         ; SEND OUT 0V
            JMP     LEDCTL
LEDOFF:     MVI     E,0FFH      ; SEND OUT 5V
LEDCTL:     MVI     C,DACSRV    ; D/A SERVICE
            CALL    MOS
            RET
```

Display Controller Software Description

The program will be described from the lowest level subroutine to the main routine.

LEDON, LEDOFF
The subroutine LEDON turns on the selected display by sending 0V from the D/A into the 74LS240 enables and LEDOFF turns them off by sending 5V.

FLSHDG
This CALLs LEDON, goes into a delay loop and then CALLs LEDOFF. This causes the display selected by bit 7 to display for the period of time of the delay.

BIN7SG
This converts the number in the accumulator (A), which is in the range of 0 to F hex, to its corresponding binary pattern which will be used by another routine to illuminate the desired display segments. Since each element of a digit is controlled by bits 0 to 6 the bit pattern sent to the output port will form specific patterns. The table TAB7SG used by this routine has these bit patterns for digits 0 to F.

HEXOUT
This displays the hex value of the B register on the displays. This routine must be called repeatedly in order for the data to appear to be shown continuously, since it works on the principle of persistence of vision. The upper 4 bits of B are masked off leaving only the lower 4 bits which are converted to the

appropriate binary pattern using BIN7SG and this pattern is sent to the output port. Since the patterns received from BIN7SG always have bit 7 cleared, this will turn on the digit on the right when FLSHDG is called. To display the left digit, the lower 4 bits are masked off of B and the upper 4 are moved to the lower 4 bit positions. This value is converted using BIN7SG, bit 7 of the result is set to 1, and it is sent to the output port. This time when FLSHDG is called, the left digit will be displayed since bit 7 is set. The main loop of this first example gets its input from the DIP switches, copies the value to B, CALLs HEXOUT and loops back to read the DIP switches again.

Using the Program

Build the circuit and then check your work. Now load the following program into memory and run it. With all the DIP switches in the ON position the port will input 00 and this should be shown on the displays. The binary value input to the DIP switches will be shown in hex on the displays (refer to the section at the beginning of this manual which discusses binary to hex conversion). Set the DIP switches so one digit is different than the other.

It appears that both digits are showing at the same time. To show what is really happening, we can increase the delay in FLSHDG so we can see what is really happening. Change the byte at FF4B from 00 to FF and run the program again. The displays can now be seen alternating left to right with each change in bit 7. Note that the PRIMER's digital output LEDs reflect the data sent to the output port (output bits of 0 turn on these LEDs). Watch the binary pattern on bits 6 to 0 as the digits change.

Move the DIP switches to the off position so that "FF" is displayed (this guarantees that none of the inputs are being pulled low), stop the program and change the byte at FF4B back to 00 again.

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION	
FF01	D3	IN	12
FF02	12		
FF03	47	MOV	B,A
FF04	CD	CALL	FF0A
FF05	0A		
FF06	FF		
FF07	C3	JMP	FF01
FF08	01		
FF09	FF		
FF0A	78	MOV	A,B
FF0B	E6	ANI	0F
FF0C	0F		
FF0D	CD	CALL	FF27
FF0E	27		
FF0F	FF		
FF10	D3	OUT	11
FF11	11		
FF12	CD	CALL	FF44
FF13	44		
FF14	FF		
FF15	78	MOV	A,B
FF16	E6	ANI	F0
FF17	F0		
FF18	0F	RRC	
FF19	0F	RRC	
FF1A	0F	RRC	
FF1B	0F	RRC	
FF1C	CD	CALL	FF27

ADDRESS	DATA	DESCRIPTION	
FF1D	27		
FF1E	FF		
FF1F	F6	ORI	80
FF20	80		
FF21	D3	OUT	11
FF22	11		
FF23	CD	CALL	FF44
FF24	44		
FF25	FF		
FF26	C9	RET	
FF27	E5	PUSH	H
FF28	D5	PUSH	D
FF29	11	LXI	D,FF34
FF2A	34		
FF2B	FF		
FF2C	26	MVI	H,00
FF2D	00		
FF2E	6F	MOV	L,A
FF2F	19	DAD	D
FF30	7E	MOV	A,M
FF31	D1	POP	D
FF32	E1	POP	H
FF33	C9	RET	
FF35	79	(PATTERN FOR "1")	
FF36	24	(PATTERN FOR "2")	
FF37	30	(PATTERN FOR "3")	

Continued on next page…

ADDRESS	DATA	DESCRIPTION	ADDRESS	DATA	DESCRIPTION
FF38	19	(PATTERN FOR "4")	FF4F	C2	JNZ FF4C
FF39	12	(PATTERN FOR "5")	FF50	4C	
FF3A	02	(PATTERN FOR "6")	FF51	FF	
FF3B	78	(PATTERN FOR "7")	FF52	CD	CALL FF5D
FF3D	18	(PATTERN FOR "9")	FF53	5D	
FF3E	08	(PATTERN FOR "A")	FF54	FF	
FF3F	03	(PATTERN FOR "B")	FF55	F1	POP PSW
FF40	46	(PATTERN FOR "C")	FF56	D1	POP D
FF41	21	(PATTERN FOR "D")	FF57	C9	RET
FF42	06	(PATTERN FOR "E")	FF58	1E	MVI E,00
FF43	0E	(PATTERN FOR "F")	FF59	00	
FF44	D5	PUSH D	FF5A	C3	JMP FF5F
FF45	F5	PUSH PSW	FF5B	5F	
FF46	CD	CALL FF58	FF5C	FF	
FF47	58		FF5D	1E	MVI E,FF
FF48	FF		FF5E	FF	
FF49	11	LXI D,00FF	FF5F	0E	MVI C,0E
FF4A	FF		FF60	0E	
FF4B	00		FF61	CD	CALL 1000
FF4C	1B	DCX D	FF62	00	
FF4D	7A	MOV A,D	FF63	10	
FF4E	B3	ORA E	FF64	C9	RET

Scanning the Keypad

To read a 4 by 4 matrix keypad we need 4 inputs and 4 outputs. The 4 inputs will check for a key pressed in one of the 4 columns in the current row selected by the 4 outputs. Since all of the outputs are currently being used, where do we get 4 more? We will use the same ones used for the displays but we will only use them while the displays are off (this is why we needed the circuitry to turn off both displays).

The subroutine KEYSCN (shown below), which will be added to the previous program, will be CALLed while the digits are off so that the changes in the output port will not be visible. When a key is pressed, the routine will modify the B register by shifting it left 4 bits and putting the binary value of the key into the lower 4 bits.

When KEYSCN is CALLed, output bits 0 to 3 are set to 0 to select all 4 rows at once. When the input port is read and all of the lower 4 bits are 1, this indicates no key is pressed and the routine is exited without changing B. If any of the lower 4 bits are 0 this indicates a key has been pressed. The routine then selects 1 row at a time (by setting 1 of the output bits to 0 and the others to 1) until the input port reads a 0 on any of the lower 4 bits. When this happens, the row is found, and the column is found by finding which input port bit was 0. When the row and column is found it is translated to a value from 0 to F hex. The B register is shifted 4 bits to the left and this new value is put in the lower 4 bits and the routine exits.

There is another feature in KEYSCN which keeps a key that is being held closed from modifying the B register more than 1 time. When a key is pressed, the H register is loaded with a value which defines the minimum number of times KEYSCN must be CALLed while no key is pressed before it will recognize another key press. For example, when a key is pressed, B is modified by the new key value and H is loaded with 20 hex before exiting KEYSCN. On the next entry to KEYSCN the keypad will be examined to see if a key has been pressed and if one is pressed, H is not decremented and the routine is exited without changing B. If no keys are being pressed, H is decremented and the routine is exited without changing B. If no keys are pressed for 32 (20 hex) CALLs of KEYSCN then H will be 0 and any key pressed after this time will affect the B register, and again, H will be loaded with 20 hex.

The assembly language code is listed below:

```
;
; This routine checks for a key pressed and if there is one, register B
; is shifted left one nibble and the key value is put in the low nibble.
; The subsequent CALLs after a CALL that affected B, will not affect B
; again until no key has been pressed for 20 CALLs and then a key is
; pressed again. This prevents a single key press from being
; interpreted as more than one.
;
; On entry and exit: H=debounce counter
;
DBOUNCE     EQU     20          ; NUMBER OF CALLs FOLLOWING A KEY PRESS
KEYSCN:     XRA     A           ; A=0
            OUT     OPORT       ; SELECT ALL 4 ROWS
            IN      IPORT       ; READ ALL 4 ROWS OF KEYPAD
            ANI     0FH         ; MASK OFF UPPER 4 BITS
            CPI     0FH         ; IF 0FH THEN NO KEYS PRESSED
            JNZ     KEYSC1      ; SKIP IF KEY READY

            ; NO KEY PRESSED, SO DEC. THE DEBOUNCE (IF>0) AND EXIT
            INR     H
            DCR     H           ; IS DEBOUNCE 0?
            RZ                  ; RETURN IF YES
            DCR     H           ; DEC ONCE MORE
            RET

KEYSC1:     INR     H
            DCR     H
            RNZ                 ; IF DEBOUNCE <> 0 EXIT

            ; SCAN FOR SPECIFIC ROW
            PUSH    D
            MVI     E,01111111B ; ROW SCAN VALUE (WILL BE ROTATED)
            MVI     D,-4        ; ROW ADDER (+4=0)

KEYSC2:     MOV     A,E         ; GET ROW SCAN VALUE
            RLC                 ; ROTATE IT
            OUT     OPORT       ; SEND ROW SCAN TO OUTPUT PORT
            MOV     E,A         ; SAVE BACK NEW ROW SCAN

            MOV     A,D         ; GET ROW ADDER
            ADI     4           ; INC ROW ADDER BY 4
            MOV     D,A         ; SAVE IT
            IN      IPORT       ; SEE IF THIS ROW HAS CHAR READY
            ANI     0FH         ; MASK OFF UPPER
            CPI     0FH
            JZ      KEYSC2      ; LOOP TILL <> 0FH

            ; FIND WHAT COL. IT'S IN
            MVI     L,0FFH      ; SET SO INR WILL MAKE 0
KEYPD1:     INR     L
            RRC
            JC      KEYPD1      ; LOOP TILL NO CY
            ; NOW ADD COL. TO ROW ADDER
            MOV     A,D         ; GET ROW ADDER
            ADD     L
```

```
MOV      L,A         ; L IS THE KEY PRESSED (0 TO F HEX)
; SHIFT B LEFT 1 NIBBLE AND PUT L IN LOWER NIBBLE
MOV      A,B         ; SHIFT B
ADD      A
ADD      A
ADD      A
ADD      A           ; THIS SHIFTS LEFT PADDING 0's
ADD      L           ; PUT L IN LOWER NIBBLE
MOV      B,A         ; NEW B REG

MVI      H,DBOUNCE   ; DEBOUNCE VAL. (NO KEYS ACCEPTED TILL 0)
POP      D
RET
```

Using the Program

The previous program will be modified slightly (assuming it is still in memory) by putting CALL KEYSCN in the program in place of IN IPORT, MOV B,A and a new subroutine will be added at the end. (Pay close attention to the addresses when entering the following program, since there is a skip in sequence of the addresses after the first three.) When you run the program you should see the key you press on the right display and the digit that was there before, moved to the left display. As you have just seen demonstrated in this application, multiplexing allows you to greatly extend the capabilities of an output port.

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION
FF01	CD	CALL FF65
FF02	65	
FF03	FF	
:	:	
:	:	
FF65	AF	XRA A
FF66	D3	OUT 11
FF67	11	
FF68	DB	IN 12
FF69	12	
FF6A	E6	ANI 0F
FF6B	0F	
FF6C	FE	CPI 0F
FF6D	0F	
FF6E	C2	JNZ FF76
FF6F	76	
FF70	FF	
FF71	24	INR H
FF72	25	DCR H
FF73	C8	RZ
FF74	25	DCR H
FF75	C9	RET
FF76	24	INR H
FF77	25	DCR H
FF78	C0	RNZ
FF79	D5	PUSH D
FF7A	1E	MVI E,7F
FF7B	7F	
FF7C	16	MVI D,FC
FF7D	FC	
FF7E	7B	MOV A,E

ADDRESS	DATA	DESCRIPTION
FF7F	07	RLC
FF80	D3	OUT 11
FF81	11	
FF82	5F	MOV E,A
FF83	7A	MOV A,D
FF84	C6	ADI 04
FF85	04	
FF86	57	MOV D,A
FF87	DB	IN 12
FF88	12	
FF89	E6	ANI 0F
FF8A	0F	
FF8B	FE	CPI 0F
FF8C	0F	
FF8D	CA	JZ FF7E
FF8E	7E	
FF8F	FF	
FF90	2E	MVI L,FF
FF91	FF	
FF92	2C	INR L
FF93	0F	RRC
FF94	DA	JC FF92
FF95	92	
FF96	FF	
FF97	7A	MOV A,D
FF98	85	ADD L
FF99	6F	MOV L,A
FF9A	78	MOV A,B
FF9B	87	ADD A

Continued on next page…

ADDRESS	DATA	DESCRIPTION	
FF9C	87	ADD	A
FF9D	87	ADD	A
FF9E	87	ADD	A
FF9F	85	ADD	L
FFA0	47	MOV	B,A

ADDRESS	DATA	DESCRIPTION	
FFA1	26	MVI	H,14
FFA2	14		
FFA3	D1	POP	D
FFA4	C9	RET	

Application 7: Controlling an LCD Module

Purpose

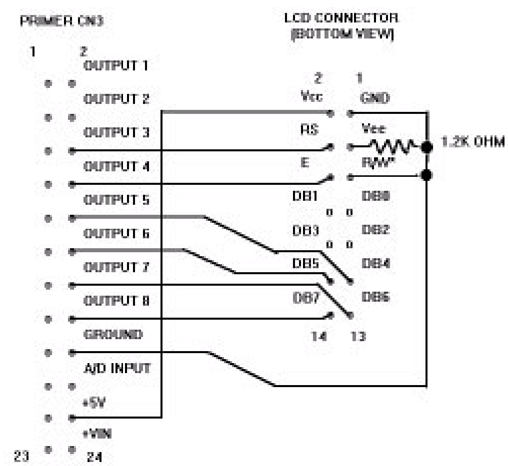
To demonstrate writing characters and cursor positioning on an LCD Module display.

Discussion

There are many LCD Module display manufacturers and most use the same 14 pin dual row header interface and the same controller chip, the HD44780. These modules display characters only, not graphics (with the exception that you can simulate graphics by dynamically defining your own characters). You may find these displays in surplus catalogs, or parts catalogs such as DIGI-KEY. Some example parts are:

DIGI-KEY Part.	Description (Call 1-800-DIGI-KEY)
OP116-ND	OPTREX 16x1 standard LCD dot matrix module
VT216-ND	Varitronix Ltd 16x2 standard LCD dot matrix module

The HD44780 controller has two registers: one for data and one for commands. The data register allows you to write characters to the display, define your own characters and read display memory. The command register allows writing of several commands relating to display control and initialization and also reading the controller's status and address counter. In the interest of simplicity we will write to the controller registers in this application.



The controller can transfer data in 8 or 4 bit mode, so we will use it in 4 bit mode since we have only 8 output ports and we need at least 4 to transfer data (DB4 to DB7) and 2 for the control lines (RS and E).

The assembly language code is listed below:

```
;
; LCD DRIVER CODE
;
OPORT    EQU      11H            ; OUTPUT PORT
IPORT    EQU      12H            ; INPUT PORT
KEYIN    EQU      0BH            ; SERVICE FOR READING KEYPAD
MOS      EQU      1000H          ; MOS CALL ADDRESS

;
; OPORT BITS ARE DEFINED AS FOLLOWS:
; 7 6 5 4 3 2 1 0
; DB7 DB6 DB5 DB4 E RS (not used)
;

         ORG      0FF01H
         MVI      A,11110011B    ; RS, E, = 0.
         OUT      OPORT

         ; RESET CODE
         CALL     DELAY
```

```
         CALL     DELAY
         MVI      A,30H
         CALL     DLNOUT
         CALL     DLNOUT
         CALL     DLNOUT

         ; INIT CODE
         MVI      A,00100000B    ; SET 4 BIT MODE
         CALL     DLNOUT

         MVI      A,00101000B    ; SET 4 BIT, 2 LINE, 5 BY 7 DOTS
         CALL     OUTCMD
         MVI      A,00001000B    ; DISPLAY OFF
         CALL     OUTCMD
         MVI      A,00000001B    ; DISPLAY ON
         CALL     OUTCMD
         MVI      A,00001110B    ; TURN ON DISPLAY, CURSOR, AND BLINK.
         CALL     OUTCMD
         MVI      A,00000110B    ; ENTRY MODE SET. INC. W/CURSOR MOVEMENT
         CALL     OUTCMD

         LXI      H,TSTSTR
         CALL     SHWSTR

LOOP:    NOP
         NOP
         NOP
         NOP
         NOP                     ; THESE ARE PLACE HOLDERS

         MVI      C,KEYIN
         CALL     MOS            ; GET A KEY
         MVI      A,'0'
         ADD      L              ; CONVERT 0 TO 9 IN L TO ASCII
         CALL     OUTDTA         ; DISPLAY THE CHAR
         JMP      LOOP

TSTSTR:  DB       'The Primer.',0

;
; Show the string pointed to by HL. When 0 is encountered the program exits
; returning HL pointing to the byte after the 0.
;
SHWSTR:  MOV      A,M            ; READ STRING
         INX      H              ; CHANGE POINTER
         ORA      A              ; SEE IF A=0
         RZ                      ; EXIT IF END OF STRING
         CALL     OUTDTA         ; DISPLAY CHARACTER
         JMP      SHWSTR

;
; Send A to the LCD with RS=1, high nibble first and low second.
;
OUTDTA:  MVI      E,0100B        ; SET RS
         JMP      OBYT1
;
; Send A to the LCD with RS=0, high nibble first and low second.
```

```
;
OUTCMD:  MVI      E,0              ; RS=0
OBYT1:   MOV      B,A              ; SAVE IN B
         ANI      0F0H             ; MASK OFF LOW NIBBLE
         ORA      E                ; MAYBE MODIFY RS
         CALL     DLNOUT           ; SEND IT
         MOV      A,B
         ADD      A
         ADD      A
         ADD      A
         ADD      A                ; LOWER IS MOVED TO UPPER, PADDING 0'S
         ORA      E                ; MAYBE MODIFY RS
         CALL     DLNOUT
         RET
;
; This delays and falls through to OUTNIB
;
DLNOUT:  CALL     DELAY

;
; Send data in A to the LCD. Assumes bits 0 to 3 have been properly set.
;
OUTNIB:  PUSH     PSW
         ANI      11110111B        ; CLEAR E
         OUT      OPORT            ; SEND NIBBLE
         ORI      1000B            ; SET E BIT
         OUT      OPORT
         ANI      11110111B        ; CLEAR E BIT
         OUT      OPORT
         POP      PSW
         RET

;
; 5 ms time delay for 8085 is 24 t states
;
DELAY:   PUSH     PSW              ; approx 5 ms for 3.072 MHZ clock
         PUSH     H
         LXI      H,641
DLAY2:   DCX      H                ; 6 T STATES
         MOV      A,H              ; 4 T STATES
         ORA      L                ; 4 T STATES
         JNZ      DLAY2            ; 10 T STATES
         POP      H
         POP      PSW
         RET
```

Program Description

According to the schematic, the output port controls the LCD and the port bits are connected as follows:

```
output port bits: 7 6 5 4 3 2 1 0
LCD header pins: DB7 DB6 DB5 DB4 E RS (not used)
```

The routine OUTNIB assumes the upper nibble of A has the value you want to output and bit 2 (RS) is set to 0 for a command or 1 for data. This value is output first with bit 3 (E) low, then high, then low again. The E input when brought high momentarily causes the data input to RS and DB4 through DB7 to be accepted by the LCD controller. DLNOUT works the same except a 5 ms delay (provided by DELAY) occurs before executing OUTNIB.

DELAY is called because the method we used to interface to the LCD Module prevents us from reading the LCD module. This in turn prevents us from reading the busy flag which tells us the LCD controller is busy executing a command and cannot receive another yet. DELAY gets us around this problem because it takes longer to execute than any of the LCD controller's instructions insuring that the LCD will not be busy by the time it is finished. In the initialization section some longer delays are needed, so DELAY is called repeatedly.

OUTCMD and OUTDTA use the same core routine but they select RS of 0 and 1 respectively. This core routine takes the byte in A and breaks it into two nibbles and sends them to DLNOUT (high nibble first).

The main routine does the hardware reset for the HD44780, followed by the display mode setup. Then SHWSTR sends the ASCII string pointed to by HL to the display via OUTDTA, and then the MOS subroutine KEYIN is called to get a key from the keypad and the key is translated to ASCII and sent to the display (via OUTDTA) and then it loops back to get another key.

Connect Primer connector CN3 to the LCD according to the schematic and then enter the following program. When you run the program "The Primer._" should be shown on the display and when you press one of keys "0" to "9" they will be shown on the display, with each new character displayed to the right of the previous.

Eventually if you press the keys enough times you will eventually run out of display area. The characters are now being stored in an area that is not being displayed. If you have a 2 line display and you send enough characters, they will start showing up on the second line and after more are sent they will eventually show up on the first line.

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION
FF01	3E	MVI A,F3
FF02	F3	
FF03	D3	OUT 11
FF04	11	
FF05	CD	CALL FF8D
FF06	8D	
FF07	FF	
FF08	CD	CALL FF8D
FF09	8D	
FF0A	FF	
FF0B	3E	MVI A,30
FF0C	30	
FF0D	CD	CALL FF7B
FF0E	7B	
FF0F	FF	
FF10	CD	CALL FF7B
FF11	7B	
FF12	FF	
FF13	CD	CALL FF7B
FF14	7B	
FF15	FF	
FF16	3E	MVI A,20
FF17	20	
FF18	CD	CALL FF7B
FF19	7B	

ADDRESS	DATA	DESCRIPTION
FF1A	FF	
FF1B	3E	MVI A,28
FF1C	28	
FF1D	CD	CALL FF68
FF1E	68	
FF1F	FF	
FF20	3E	MVI A,08
FF21	08	
FF22	CD	CALL FF68
FF23	68	
FF24	FF	
FF25	3E	MVI A,01
FF26	01	
FF27	CD	CALL FF68
FF28	68	
FF29	FF	
FF2A	3E	MVI A,0E
FF2B	0E	
FF2C	CD	CALL FF68
FF2D	68	
FF2E	FF	
FF2F	3E	MVI A,06
FF30	06	

Continued on next page…

ADDRESS	DATA	DESCRIPTION
FF31	CD	CALL FF68
FF32	68	
FF33	FF	
FF34	21	LXI H,FF4D
FF35	4D	
FF36	FF	
FF37	CD	CALL FF59
FF38	59	
FF39	FF	
FF3A	00	NOP
FF3B	00	NOP
FF3C	00	NOP
FF3D	00	NOP
FF3E	00	NOP
FF3F	0E	MVI C,0B
FF40	0B	
FF41	CD	CALL 1000
FF42	00	
FF43	10	
FF44	3E	MVI A,30
FF45	30	
FF46	85	ADD L
FF47	CD	CALL FF63
FF48	63	
FF49	FF	
FF4A	C3	JMP FF3A
FF4B	3A	
FF4C	FF	
FF4D	54	"T"
FF4E	68	"h"
FF4F	65	"e"
FF50	20	" "
FF51	50	"P"
FF52	72	"r"
FF53	69	"i"
FF54	6D	"m"
FF55	65	"e"
FF56	72	"r"
FF57	2E	"."
FF58	00	(end marker)
FF59	7E	MOV A,M
FF5A	23	INX H
FF5B	B7	ORA A
FF5C	C8	RZ
FF5D	CD	CALL FF63
FF5E	63	
FF5F	FF	
FF60	C3	JMP FF59
FF61	59	
FF62	FF	
FF63	1E	MVI E,04
FF64	04	
FF65	C3	JMP FF6A

ADDRESS	DATA	DESCRIPTION
FF66	6A	
FF67	FF	
FF68	1E	MVI E,00
FF69	00	
FF6A	47	MOV B,A
FF6B	E6	ANI F0
FF6C	F0	
FF6D	B3	ORA E
FF6E	CD	CALL FF7B
FF6F	7B	
FF70	FF	
FF71	78	MOV A,B
FF72	87	ADD A
FF73	87	ADD A
FF74	87	ADD A
FF75	87	ADD A
FF76	B3	ORA E
FF77	CD	CALL FF7B
FF78	7B	
FF79	FF	
FF7A	C9	RET
FF7B	CD	CALL FF8D
FF7C	8D	
FF7D	FF	
FF7E	F5	PUSH PSW
FF7F	E6	ANI F7
FF80	F7	
FF81	D3	OUT 11
FF82	11	
FF83	F6	ORI 08
FF84	08	
FF85	D3	OUT 11
FF86	11	
FF87	E6	ANI F7
FF88	F7	
FF89	D3	OUT 11
FF8A	11	
FF8B	F1	POP PSW
FF8C	C9	RET
FF8D	F5	PUSH PSW
FF8E	E5	PUSH H
FF8F	21	LXI H,0281
FF90	81	
FF91	02	
FF92	2B	DCX H
FF93	7C	MOV A,H
FF94	B5	ORA L
FF95	C2	JNZ FF92
FF96	92	
FF97	FF	
FF98	E1	POP H
FF99	F1	POP PSW
FF9A	C9	RET

In the next example we will modify the program to use the Set DD RAM Address command which will in effect allow us to control the cursor position. Modify the following addresses and run the program. You will see that each key typed will show up on the screen in the same place even though it is still automatically incrementing the cursor position. This is because the address is set for that cursor position after the cursor has been incremented.

You may want to experiment with different cursor positions. If you have a 2 line display, you can move the cursor to line 2 by sending 10000000b + 40h (C0h) to OUTCMD, where 10000000b is the command for Set DD RAM Address and 40h is the offset for line 2.

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION
FF3A	3E	MVI A,8B
FF3B	8B	
FF3C	CD	CALL FF68
FF3D	68	
FF3E	FF	

Application 8: Capacitance Meter

Purpose

This application shows how to use the PRIMER as a capacitance meter.

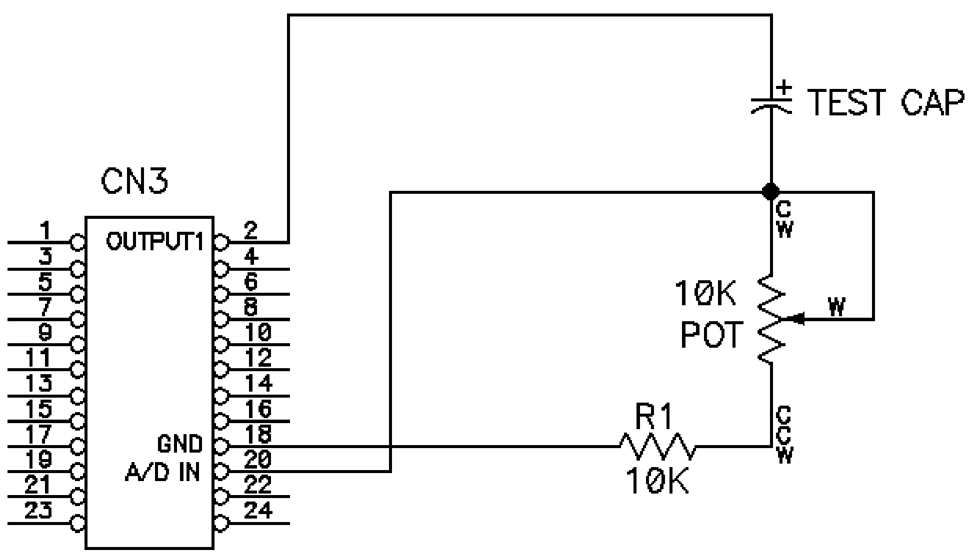
Discussion

This application is an example of how the PRIMER can be used as a useful piece of electronic test equipment. The Capacitance Meter application can be used to accurately measure capacitors ranging from 0.01 µF to 220 µF.

The parts required are minimal. Items needed are:

Qty.	Description
1	10 KΩ multi-turn potentiometer
1	10 KΩ 1/4 watt resistor
1	one capacitor of a known value in the range of 1 to 100 µF (calibration cap)
Many	Several capacitors, for testing, in the range of 0.01 µF to 300 µF
1	Breadboard

The circuit is very simple. Follow the schematic below to assemble the circuit.



Circuit Description:

The PRIMER uses its on-board D/A converter, analog input comparator, OUTPUT1 (digital output line 1), and the timer within the 8155 to measure capacitance. The capacitor is connected in series with R1. The open end of the capacitor is then tied to the OUTPUT1 pin of CN3 and the open end of R1 is tied to ground. The D/A output of the PRIMER is already internally tied to the non-inverting side of the op-amp comparator while the capacitor-R1 connection is tied to the inverting side via the A/D IN pin on CN3. When the program first starts, the D/A is set slightly above ground potential and OUTPUT1 is set LOW. The capacitor now discharges through R1 and the potentiometer. The program waits for the user to press a key at the keypad and when it is pressed, the program then starts the timer and sets OUTPUT1 HI which starts the capacitor charging. The timer is driven by a 307.2 KHz input Clock. The timer works by loading a “count” value into a register within the timer. The timer then decrements this value after each complete input clock cycle. When the value reaches 0, the timer generates an output pulse that is detected by the 8085’s RST 7.5 interrupt then the timer automatically reloads the register with the “count” value and the process starts all over again. By increasing the value in the “count” register the pulse rate can be slowed down and vise-versa. The Capacitor Meter program uses the timer as the time-base by counting how many pulses are generated by the timer while the capacitor is charging (using the 8085’s DE register pair). The larger the capacitor, the longer the charge time, therefore the more pulses will be generated. The voltage across the resistor is near VCC when OUTPUT1 first goes high, then ramps down as the capacitor charges. When the voltage falls below the D/A voltage threshold, the comparator output goes high, and the timer is stopped. The current pulse count in DE is then converted to decimal and displayed on the LED display and then the decimal point is placed in the proper place in the number.

Theory of Operation

The Capacitor Meter program works by measuring the time required to charge the capacitor through a resistor. The time-base is generated by the timer within the 8155. The Capacitor Meter program has 2 capacitance ranges from which to choose The low range can measure capacitor values up to 9.999 μF in 0.001 μF increments while the high range can measure values up to 999.9 μF in 0.1 μF increments. Two scales were chosen to provide good resolution while reading small capacitors but also have the ability to measure large caps. The scale is determined by the “count” value loaded into the 8155 timer. A value of 10 is loaded in the “count” register for low range and a value of 1000 for the high range . Once the capacitor is charged, the pulse count is displayed on the LED display in decimal. A decimal point is then placed on the LED display in the “10's” place for high range or in the “1000's” place for low range. So the actual value written to the display for a 1 μF capacitor measured in low scale would be “1000". Once the decimal point is added it looks like “1.000".

The equation for capacitor charge time in an RC (resister, capacitor) circuit is:

$$T = R \cdot C$$

Where:

T = Time in Seconds
R = Resistance in Ohms
C = Capacitance in Farads

Note that this simple formula is not sufficient to predict the value for *R* since in this application the capacitor is not fully charged (because of limits in the D/A’s resolution) and because of the parallel resistance of the circuitry on the PRIMER board, and due to the factor of ESR (Equivalent Series Resistance) of the capacitor. However, when *R* is known, *T* can be predicted based upon the known value of *C*. After the calibration procedure (discussed later) is performed, the value for *R* will be fixed, which will give a charge time directly proportional to the capacitance.

The 8155 timer is decremented every 1/307200 seconds (or 3.26 μs). Since the time-base in low range is (10*3.26 μs or 32.6 μs) and each time interval in this range represents 0.001 μF, the resistance needs to be adjusted so that, for example, a 1 μF capacitor charges up to the threshold in 1000*32.6 μs. When the resistance is precisely adjusted for the low range, it is also suitable for the high range. Since The time-base in the high range is (1000*3.26 μs or 3.26 ms) and each time interval in this range represents 0.1 μF, A 100 μF capacitor will take 1000*3.26 ms or 3.26 seconds to charge to the threshold in a properly calibrated circuit.

Using the Program

The assembly language code is listed below:

```
;           CAPACITANCE METER
            .8085
DIPSW       EQU   12H             ; ADDRESS OF PORT A (DIPSWITCH)
P_OUT       EQU   11H             ; ADDRESS OF PORT B (OUTPUT PORT)
P_8155      EQU   10H             ; ADDRESS OF 8155 CONTROL REGISTER
P_CNTLO     EQU   14H             ; ADDRESS OF LO BYTE OF COUNTER
P_CNTHI     EQU   15H             ; ADDRESS OF HI BYTE OF COUNTER
TMRSTRT     EQU   0CDH            ; START TIMER COMMAND
TMRSTOP     EQU   8DH             ; STOP TIMER COMMAND
ADCVAL      EQU   01H             ; VALUE OF 1 TO D/A
TMRMODE     EQU   0C0H            ; SINGLE PULSE AND RELOAD
DSPORT      EQU   40H             ; ADDRESS OF LED DISPLAY DATA
DSPCMD      EQU   41H             ; ADDRESS OF LED DISPLAY COMMAND REGISTER
MOS         EQU   1000H           ; MOS SERVICE ACCESS
DACOUT      EQU   0EH             ; DACOUT SERVICE
LEDDEC      EQU   13H             ; LEDDEC SERVICE
KEYSTAT     EQU   16H             ; KEYSTAT SERVICE

            ORG   0FF01H          ; ORIGIN OF MEM IN 8155

START:      MVI   E,ADCVAL        ; SET D/A TO LOW V
            MVI   C,DACOUT        ; MOS SERVICE #
            CALL  MOS

            MVI   A,TMRSTOP       ; STOP TIMER
            OUT   P_8155

            LXI   D,0000H         ; CLR D,E (PUT 0'S IN LED DISPLAY)
            MVI   C,LEDDEC        ; MOS SERVICE #
            CALL  MOS

            MVI   A,80H           ; "WRITE COMMAND" FOR DIGIT 0
            OUT   DSPCMD

            MVI   A,00010111B     ; WRITE "F" TO DIGIT 0
            OUT   DSPORT

            MVI   A,81H           ; "WRITE COMMAND" FOR DIGIT 1
            OUT   DSPCMD

            MVI   A,11000001B     ; WRITE PATTERN FOR "u" TO DIGIT 1
            OUT   DSPORT

            ; begin discharging cap
            XRA   A               ; CLEAR ACC
            OUT   P_OUT           ; SET PORT A LO

            ; Wait for keypad key press and update decimal point setting
WAIT:       IN    DIPSW           ; GET SW0 SETTING
            ANI   01              ; MASK OFF OTHER SWITCHES

            MVI   C,5             ; DECIMAL DIG 5
            MOV   B,A
```

```
            CALL  DECPNT          ; PLACES THE DECIMAL POINT
            XRI   00000001B       ; COMPLIMENT SW SETTING
            MOV   B,A
            MVI   C,3
            CALL  DECPNT
            MOV   B,A             ; SAVE SWITCH VAL

            MVI   C,KEYSTAT       ; MOS SERVICE #
            CALL  MOS             ; GET KEYPAD STATUS

            MOV   A,H
            RAR                   ; IF KEY NOT PRESSED
            JNC   WAIT            ; THEN WAIT

            ; Key pressed. Now charge the cap
            MVI   A,0FFH          ; SET OUTPUT1 HIGH
            OUT                   P_OUT

            ; Load timer with timebase value based on DIP switch setting
            MOV   A,B             ; IF DIPSWITCH1 IS ON
            RAR
            JNC   HI              ; THEN GOTO HI

LO:         MVI   A,0E8H          ; LOAD TIMER W/ 1000 DECIMAL
            OUT   P_CNTLO
            MVI   A,0C3H
            OUT   P_CNTHI
            JMP   GO

HI:         MVI   A,0AH           ; LOAD TIMER W/ 10 DECIMAL
            OUT   P_CNTLO
            MVI   A,0C0H
            OUT   P_CNTHI

GO:         MVI   A,TMRSTRT       ; START TIMER
            OUT   P_8155

CAPCNT:     MVI   A,1FH           ; CLEAR 7.5 INT (TIMER INTERRUPT)
            SIM                   ; SET INTERUPT MASK

            ; Poll the SID line till it goes high
            ; (when the comparator outputs a high)
POLL2:      RIM                   ; LOAD ACC WITH INT FLG STATUS
            RAL                   ; CHECK IF SID HAS GONE HIGH
            JC    SHWCAP          ; IF SO THEN EXIT AND SHOW CAP VALUE
            RAL                   ; CHECK IF 7.5 INT IS SET
            JNC   POLL2           ; IF NOT THEN POLL TILL SET

            INX   D               ; INCREMENT DE REGISTER PAIR WITH EACH 7.5
INTERRUPT
            JMP   CAPCNT

SHWCAP:     MVI   C,LEDDEC        ; SHOW DECIMAL VALUE OF DE REGISTER PAIR
            CALL  MOS

            MOV   A,B
            MVI   C,3
```

```
            CALL  DECPNT         ; PLACE THE DECIMAL POINT
            XRI   00000001B      ; COMPLIMENT SW SETTING
            MOV   B,A
            MVI   C,5
            CALL  DECPNT

            ; the display shows the cap reading, now wait till
            ; a keypad key is pressed before going on.
WAITKY:     MVI   C,KEYSTAT      ; MOS SERVICE #
            CALL  MOS
            MOV   A,H
            RAR                  ; IF A BUTTON WAS NOT PRESSED,
            JNC   WAITKY         ; THEN POLL

            JMP   START          ; ELSE TEST ANOTHER CAP

; *******************************************************
; DECPNT:   IN: LOAD C W/ DIGIT #, LOAD B WITH A 1 OR 0
;           B=1 DEC PNT ON, B=0 DEC PNT OFF
;           OUT: NOTHING
; -------------------------------------------------------
DECPNT:     PUSH  PSW
            MOV   A,B
            RAL                  ; MOVE BIT 0 TO BIT 3 LOCATION
            RAL
            RAL
            ANI   00001000B
            MOV   B,A
            MVI   A,60H
            ADD   C              ; COMMAND TO READ DIGIT
            OUT   DSPCMD
            IN    DSPORT         ; GET SEGMENT VALUES
            STA   TEMP           ; SAVE A REG
            MVI   A,80H          ; COMMAND TO WRITE DIGIT
            ADD   C
            OUT   DSPCMD
            LDA   TEMP           ; RECALL A VALUE
            ANI   11110111B      ; TURN OFF DECIMAL POINT
            ORA   B              ; TURN ON IF SUPPOSED TO BE ON
            OUT   DSPORT         ; WRITE A TO DIGIT
            POP   PSW
            RET

TEMP        DS    1

            END
```

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION
FF01	1E	MVI E,01
FF02	01	
FF03	0E	MVI C,0E
FF04	0E	
FF05	CD	CALL 1000
FF06	00	
FF07	10	
FF08	3E	MVI A,8D
FF09	8D	
FF0A	D3	OUT 10
FF0B	10	
FF0C	11	LXI D,0000
FF0D	00	
FF0E	00	
FF0F	0E	MVI C,13
FF10	13	
FF11	CD	CALL 1000
FF12	00	
FF13	10	
FF14	3E	MVI A,80
FF15	80	
FF16	D3	OUT 41
FF17	41	
FF18	3E	MVI A,17
FF19	17	
FF1A	D3	OUT 40
FF1B	40	
FF1C	3E	MVI A,81
FF1D	81	
FF1E	D3	OUT 41
FF1F	41	
FF20	3E	MVI A,C1
FF21	C1	
FF22	D3	OUT 40
FF23	40	
FF24	AF	XRA A
FF25	D3	OUT 11
FF26	11	
FF27	DB	IN 12
FF28	12	
FF29	E6	ANI 01
FF2A	01	
FF2B	0E	MVI C,05
FF2C	05	
FF2D	47	MOV B,A
FF2E	CD	CALL FF94
FF2F	94	
FF30	FF	
FF31	EE	XRI 01
FF32	01	
FF33	47	MOV B,A
FF34	0E	MVI C,03
FF35	03	
FF36	CD	CALL FF94
FF37	94	
FF38	FF	
FF39	47	MOV B,A
FF3A	0E	MVI C,16
FF3B	16	
FF3C	CD	CALL 1000
FF3D	00	
FF3E	10	
FF3F	7C	MOV A,H
FF40	1F	RAR
FF41	D2	JNC FF27
FF42	27	
FF43	FF	
FF44	3E	MVI A,FF
FF45	FF	
FF46	D3	OUT 11
FF47	11	
FF48	78	MOV A,B
FF49	1F	RAR
FF4A	D2	JNC FF58
FF4B	58	
FF4C	FF	
FF4D	3E	MVI A,E8
FF4E	E8	
FF4F	D3	OUT 14
FF50	14	
FF51	3E	MVI A,C3
FF52	C3	
FF53	D3	OUT 15
FF54	15	
FF55	C3	JMP FF60
FF56	60	
FF57	FF	
FF58	3E	MVI A,0A
FF59	0A	
FF5A	D3	OUT 14
FF5B	14	
FF5C	3E	MVI A,C0
FF5D	C0	
FF5E	D3	OUT 15
FF5F	15	
FF60	3E	MVI A,CD
FF61	CD	
FF62	D3	OUT 10
FF63	10	
FF64	3E	MVI A,1F
FF65	1F	
FF66	30	SIM
FF67	20	RIM
FF68	17	RAL
FF69	DA	JC FF74
FF6A	74	

Continued on next page…

ADDRESS	DATA	DESCRIPTION
FF6B	FF	
FF6C	17	RAL
FF6D	D2	JNC FF67
FF6E	67	
FF6F	FF	
FF70	13	INX D
FF71	C3	JMP FF64
FF72	64	
FF73	FF	
FF74	0E	MVI C,13
FF75	13	
FF76	CD	CALL 1000
FF77	00	
FF78	10	
FF79	78	MOV A,B
FF7A	0E	MVI C,03
FF7B	03	
FF7C	CD	CALL FF94
FF7D	94	
FF7E	FF	
FF7F	EE	XRI 01
FF80	01	
FF81	47	MOV B,A
FF82	0E	MVI C,05
FF83	05	
FF84	CD	CALL FF94
FF85	94	
FF86	FF	
FF87	0E	MVI C,16
FF88	16	
FF89	CD	CALL 1000
FF8A	00	
FF8B	10	
FF8C	7C	MOV A,H
FF8D	1F	RAR
FF8E	D2	JNC FF87
FF8F	87	

ADDRESS	DATA	DESCRIPTION
FF90	FF	
FF91	C3	JMP FF01
FF92	01	
FF93	FF	
FF94	F5	PUSH PSW
FF95	78	MOV A,B
FF96	17	RAL
FF97	17	RAL
FF98	17	RAL
FF99	E6	ANI 08
FF9A	08	
FF9B	47	MOV B,A
FF9C	3E	MVI A,60
FF9D	60	
FF9E	81	ADD C
FF9F	D3	OUT 41
FFA0	41	
FFA1	DB	IN 40
FFA2	40	
FFA3	32	STA FFB5
FFA4	B5	
FFA5	FF	
FFA6	3E	MVI A,80
FFA7	80	
FFA8	81	ADD C
FFA9	D3	OUT 41
FFAA	41	
FFAB	3A	LDA FFB5
FFAC	B5	
FFAD	FF	
FFAE	E6	ANI F7
FFAF	F7	
FFB0	B0	ORA B
FFB1	D3	OUT 40
FFB2	40	
FFB3	F1	POP PSW
FFB4	C9	RET

Calibration and Use

After loading the program, set the potentiometer for midscale and install the calibration capacitor. Press FUNC. then RUN (to enter run mode). The display should read "0000 µF" with a decimal point in the "10's" place or in the "1000's" place. Change DIPSWITCH 0 to change the decimal point position. With the decimal point in the “10's” place, the Capacitor Meter program can measure capacitor values up to 999.9 µF. With the decimal point in the “1000's” place, values up to 9.999 µF can be measured. Once the scale is chosen, press any key on the keypad and the perceived value will be shown on the display. Press another key to start the program over again. This zero’s out the numeric display and starts discharging the capacitor (indicated by LD0-LD7 being lit). For larger capacitors, such as 300 µF, you need to wait up to 10 seconds for the capacitor to fully discharge after a reading, therefore it is recommended that smaller capacitors such as 1 µF be used for calibration in the low range. Adjust the potentiometer and continue to test the calibration capacitor until an accurate reading is realized. Test several caps and record the results. Accuracies greater than 99% are possible.

NOTE: The most accurate results will be obtained when the PRIMER is powered up and the temperature allowed to stabilize over a period of 15 to 30 minutes.

Application 9: Interfacing a Stepper Motor to the PRIMER

Purpose

To show how a computer can be used to perform motion control using a stepper motor.

Goals

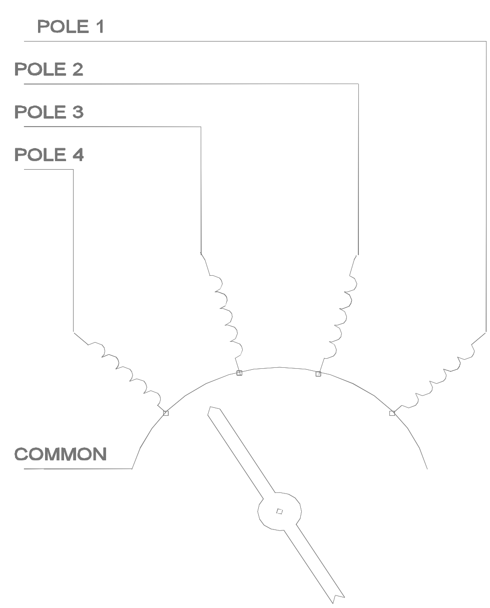
1. Build a stepper motor driver circuit.
2. Load a program that will demonstrate stepper motor control.

Materials

Qty.	Description
1	PRIMER trainer
1	breadboard
1	SM4200 4 Phase stepper motor (Jameco part #105890. Call 1-800-831-4242)
1	7404 Hex Inverter
4	2N3904 NPN Transistors
4	1N4001 Diodes
4	1 KΩ, 1/4 Watt Resistor
1	220 Ω, 1/4 Watt Resistor

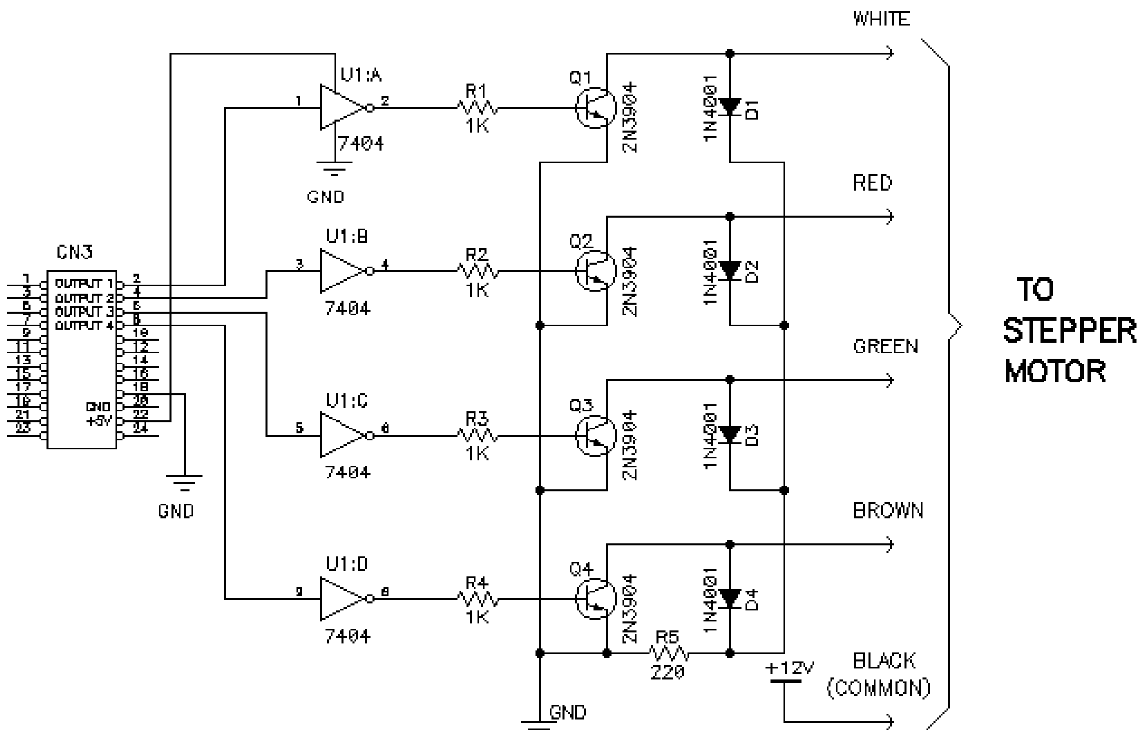
Discussion

This lab shows how the PRIMER can be used to drive a stepper motor. The diagram below shows the electrical equivalent of a 4-phase stepper motor connected to the output port of the PRIMER. When the program first starts, OUTPUT2 and OUTPUT3 are energized. The stepper is now held in position because of the magnetic force pulling the rotor between the energized poles. A step can be made by turning on OUTPUT4 while turning off OUTPUT2. This moves the rotor one increment. To move one more increment, OUTPUT1 is turned on while OUTPUT3 is turned off. To go back to the original position, the sequence would be as follows: Turn on OUTPUT3 while turning off OUTPUT1, turn on OUTPUT2 while turning off OUTPUT4.



Circuit Description and Construction

The stepper motor cannot connect directly to the output port of the PRIMER because it uses 5 volt logic levels while the stepper motor operates on 12 volts. The current demand of the stepper motor is also a problem, since computer logic supplies very low current compared to the stepper motor's needs. The solution to these problems is an interface circuit. The circuit shown in the schematic provides the necessary interface from 5 volt logic to a 12 volt source required by the stepper. Transistors Q1-Q4 provide the current and voltage amplification while diodes D1-D4 and resistor R5 provide a feedback path for the back EMF generated when the poles are de-energized. The inverters are used to convert the negative logic on the PRIMER to positive logic and to prevent the stepper from being energized when the PRIMER is reset. The interface is connected to the low nibble (4 bits) of the PRIMER output port. The driver circuit should be built on a breadboard following the schematic. Once built, a small piece of solid wire should be tightly wrapped around the shaft of the stepper motor to serve as a pointing device.



Note: The stepper motor and driver circuit are powered from a power supply separate from the PRIMER itself. This is necessary because of the large current draw and noise produce by the stepper motor.

Using the Program

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION	ADDRESS	DATA	DESCRIPTION
FF01	1E	MVI E,37	FF0A	1E	MVI E,FB
FF02	37		FF0B	FB	
FF03	16	MVI D,01	FF0C	15	DCR D
FF04	01		FF0D	CD	CALL 1000
FF05	0E	MVI C,11	FF0E	00	
FF06	11		FF0F	10	
FF07	CD	CALL 1000	FF10	3E	MVI A,33
FF08	00				
FF09	10				

Continued on next page…

ADDRESS	DATA	DESCRIPTION
FF11	33	
FF12	32	STA FFAC
FF13	AC	
FF14	FF	
FF15	AF	XRA A
FF16	32	STA FFAD
FF17	AD	
FF18	FF	
FF19	6F	MOV L,A
FF1A	47	MOV B,A
FF1B	C3	JMP FF43
FF1C	43	
FF1D	FF	
FF1E	78	MOV A,B
FF1F	32	STA FFAD
FF20	AD	
FF21	FF	
FF22	CD	CALL FF4B
FF23	4B	
FF24	FF	
FF25	3A	LDA FFAD
FF26	AD	
FF27	FF	
FF28	47	MOV B,A
FF29	16	MVI D,00
FF2A	00	
FF2B	58	MOV E,B
FF2C	0E	MVI C,13
FF2D	13	
FF2E	CD	CALL 1000
FF2F	00	
FF30	10	
FF31	7D	MOV A,L
FF32	90	SUB B
FF33	CA	JZ FF1E
FF34	1E	
FF35	FF	
FF36	DA	JC FF3F
FF37	3F	
FF38	FF	
FF39	04	INR B
FF3A	AF	XRA A
FF3B	5F	MOV E,A
FF3C	C3	JMP FF43
FF3D	43	
FF3E	FF	
FF3F	05	DCR B
FF40	AF	XRA A
FF41	3C	INR A
FF42	5F	MOV E,A
FF43	16	MVI D,64
FF44	64	
FF45	CD	CALL FF72
FF46	72	
FF47	FF	
FF48	C3	JMP FF29

ADDRESS	DATA	DESCRIPTION
FF49	29	
FF4A	FF	
FF4B	06	MVI B,02
FF4C	02	
FF4D	0E	MVI C,0B
FF4E	0B	
FF4F	CD	CALL 1000
FF50	00	
FF51	10	
FF52	7D	MOV A,L
FF53	FE	CPI 0A
FF54	0A	
FF55	D2	JNC FF4D
FF56	4D	
FF57	FF	
FF58	05	DCR B
FF59	CA	JZ FF62
FF5A	62	
FF5B	FF	
FF5C	32	STA FFAA
FF5D	AA	
FF5E	FF	
FF5F	C3	JMP FF4D
FF60	4D	
FF61	FF	
FF62	32	STA FFAB
FF63	AB	
FF64	FF	
FF65	3A	LDA FFAA
FF66	AA	
FF67	FF	
FF68	47	MOV B,A
FF69	CD	CALL FFA1
FF6A	A1	
FF6B	FF	
FF6C	3A	LDA FFAB
FF6D	AB	
FF6E	FF	
FF6F	80	ADD B
FF70	6F	MOV L,A
FF71	C9	RET
FF72	F5	PUSH PSW
FF73	C5	PUSH B
FF74	7B	MOV A,E
FF75	1F	RAR
FF76	3A	LDA FFAC
FF77	AC	
FF78	FF	
FF79	DA	JC FF80
FF7A	80	
FF7B	FF	
FF7C	0F	RRC
FF7D	C3	JMP FF81
FF7E	81	

Continued on next page…

ADDRESS	DATA	DESCRIPTION	
FF7F	FF		
FF80	07	RLC	
FF81	32	STA	FFAC
FF82	AC		
FF83	FF		
FF84	DB	IN	11
FF85	11		
FF86	E6	ANI	F0
FF87	F0		
FF88	47	MOV	B,A
FF89	3A	LDA	FFAC
FF8A	AC		
FF8B	FF		
FF8C	E6	ANI	0F
FF8D	0F		
FF8E	B0	ORA	B
FF8F	D3	OUT	11
FF90	11		
FF91	D5	PUSH	D
FF92	06	MVI	B,FF
FF93	FF		
FF94	05	DCR	B

ADDRESS	DATA	DESCRIPTION	
FF95	C2	JNZ	FF94
FF96	94		
FF97	FF		
FF98	00	NOP	
FF99	15	DCR	D
FF9A	C2	JNZ	FF92
FF9B	92		
FF9C	FF		
FF9D	D1	POP	D
FF9E	C1	POP	B
FF9F	F1	POP	PSW
FFA0	C9	RET	
FFA1	F5	PUSH	PSW
FFA2	78	MOV	A,B
FFA3	07	RLC	
FFA4	07	RLC	
FFA5	80	ADD	B
FFA6	07	RLC	
FFA7	47	MOV	B,A
FFA8	F1	POP	PSW
FFA9	C9	RET	

Once the program is started the LED display should read "0000 P0.". The "P0." Stands for "position" and "0000" indicates the relative position of the stepper referenced from its original position when the program was started (thus 0000 means it is in the same position as it was on start up). Press a two digit decimal number on the keypad and the stepper motor should move to that position with the display incrementing as the stepper moves. Once the stepper stops, enter 00 and the stepper should rotate the opposite direction with the display decrementing and finally stopping at 00. The stepper motor should now be in the exact position it was in when the program was first started.

Program Description

The subroutines are described as follows:

`DBLDECIN` - Waits for two decimal keys to be pressed then returns the decimal equivalent in the L register. The routine contains error trapping that will not allow a key greater than 9 or a control key to be accepted.

`MULTX10` - Used by `DBLDECIN` to multiply the first key press by a factor of ten. This routine may come in handy in other programs.

`STEPR` - Moves the stepper motor one step forward or backward. The speed can be controlled by changing the label `SPEED`, and the direction is controlled by the value in the E register.

The assembly language code is listed below:

```
;           STEPPER MOTOR PROG

P_IN        EQU   12H           ; ADRES OF PORT A
P_OUT       EQU   11H           ; ADRES OF PORT B
MOS         EQU   1000H         ; MOS SERVICE
KEYIN       EQU   0BH           ; VECTOR FOR KEYIN SERVICE
LEDDEC      EQU   13H           ; VECTOR FOR LEDDEC SERVICE
```

```
SPEED       EQU   20            ; STEPR MOTOR SPEED
LEDOUT      EQU   11H

            ORG   0FF01H        ; ORIGIN OF MEM IN 8155
START:
            MVI   E,00110111B   ; THE VALUE FOR "P"
            MVI   D,1
            MVI   C,LEDOUT
            CALL  MOS

            MVI   E,11111011B   ; THE VALUE FOR "O."
            DCR   D
            CALL  MOS

            MVI   A,00110011B   ; INITIALIZE STEPPER MOTOR   ;
            STA   STEP          ; STORE IN STEP
            XRA   A             ; CLR A REG
            STA   FINLPOS       ; CLR FINLPOS VARIABLE
            MOV   L,A           ; CLR L REG
            MOV   B,A           ; CLR B REG
            JMP   SKPCW         ; JUMP TO OUTPUT START POS TO STEPPER
MAIN:
            MOV   A,B           ; NEW POSITION BECOMES OLD POSITION
            STA   FINLPOS

            CALL  DBLDECIN      ; GET KEY BOARD VALUE

            LDA   FINLPOS
            MOV   B,A
STEPLUP:
            MVI   D,0           ; CLR D REG
            MOV   E,B           ; PLACE CURRENT POSITION ON LED DISPLAY
            MVI   C,LEDDEC
            CALL  MOS

            MOV   A,L           ; WHERE SUPPOSED TO BE
            SUB   B             ; - WHERE AT
            JZ    MAIN          ; IF 0 EXIT LUP AND START OVER
            JC    CW            ; IF NEG GOTO CW ELSE CCW
CCW:
            INR   B             ; INC CURENT POSITION
            XRA   A             ; CLR A REG

            MOV   E,A           ; E = 0
            JMP   SKPCW
CW:
            DCR   B             ; DEC CURRENT POS
            XRA   A             ; CLR A REG
            INR   A             ; A = 1
            MOV   E,A           ; E = 1
SKPCW:
            MVI   D,SPEED       ; SET SPEED OF STEPR
            CALL  STEPR
            JMP   STEPLUP       ; REPEAT
```

```
; ******************************************************************
; DOUBLE DECIMAL IN
; INPUT: NOTHING.
; OUTPUT: L = BINARY VALUE OF A TWO DECIMAL DIGIT INPUT FROM KEYPAD
;
; ------------------------------------------------------------------
DBLDECIN:
            MVI   B,2            ; USED AS COUNTER TO CALL KEYIN TWICE
GETPOS:
            MVI   C,KEYIN
            CALL  MOS            ; CALL KEYIN
            MOV   A,L            ; A = KEY VALUE
            CPI   10             ; IF VALUE IS > 10 ENTER AGAIN
            JNC   GETPOS
            DCR   B              ; DEC LOOP COUNTER
            JZ    LOLBLE         ; IF ZERO THEN EXIT
            STA   HIDIG          ; IF NOT THEN STORE FIRST KEYPRESS AS
            JMP   GETPOS         ; HIGH DIGIT
LOLBLE:
            STA   LODIG          ; STORE SECOND DIGIT AS LOW DIGIT
            LDA   HIDIG          ; LOAD HIGH DIG
            MOV   B,A            ; MOV TO B
            CALL  MULTX10        ; MULTIPLY IT BY TEN
            LDA   LODIG          ; LOAD LOW DIG
            ADD   B              ; ADD IT TO HI DIGIT
            MOV   L,A            ; STORE FINAL DEC VAL IN L
            RET
; ********************************************************************
; STEPR
; IN: D = SPEED. E = DIRECTION,1 = CW 0 = CCW
; OUT: NOTHING
; --------------------------------------------------------------------
STEPR:
            PUSH  PSW            ; SAVE A STATUS
            PUSH  B              ; SAVE B STATUS
            MOV   A,E            ;
            RAR
            LDA   STEP           ; LOAD STEP
            JC    LEFT           ; IF E = 1 THEN GOTO LEFT
            RRC                  ; ELSE ROTATE STEP RIGHT
            JMP   SKIP           ; SKIP NEXT INSTRUCTION
LEFT:
            RLC                  ; ROTATE STEP LEFT
SKIP:
            STA   STEP           ; STORE BACK AS STEP

            IN    P_OUT          ; MASK OFF 4 LSB OF OUTPUT PORT
            ANI   0F0H
            MOV   B,A
            LDA   STEP           ; LOAD STEP
            ANI   0FH            ; MASK OFF 4 MSB OF STEP
            ORA   B              ; OR WITH 4 LSB OF OUTPUT PORT
            OUT   P_OUT          ; OUT STEP AS 4 LSB'S AND CURRENT STATUS OF 4
                                 ; MSB'S OF OUTPUT PORT REMAIN UNCHANGED.
            PUSH  D
DELAY:
            MVI   B,0FFH         ; DELAY TO CONTROL SPEED OF STEPPER
```

```
DEL:
            DCR   B
            JNZ   DEL
            NOP
            DCR   D
            JNZ   DELAY
            POP   D
            POP   B
            POP   PSW
            RET
; ****************************************************************
; INPUT: B = VALUE TO MULT BY 10, MUST BE LESS THAN 25 DECIMAL
; ------------------------------------------------------------------
MULTX10:
            PUSH  PSW
            MOV   A,B
            RLC
            RLC
            ADD   B
            RLC
            MOV   B,A
            POP   PSW
            RET

HIDIG       DS    1
LODIG       DS    1
STEP        DS    1
FINLPOS     DS    1

            END
```

Application 10: Interfacing an 8255A PPI to the PRIMER

Purpose

To introduce the method of interfacing an I/O mapped device to the PRIMER by building a simple circuit using the 8255A PPI.

Materials

Qty.	Description
1	PRIMER trainer
1	8255A PPI Chip
1	Breadboard
2	50 pin ribbon cable female header connector
1	6 inch portion of 50 wire ribbon cable
1	7 inches of wire-wrap wire and a wire-wrapping tool
40	18 gauge jumper wires 4 to 6 inches long
1	1 KΩ 5% 1/4 W resistor
24	LED's

Introduction to the 8255A PPI

The 8255A PPI (programmable peripheral interface) is a general purpose programmable I/O device designed to use with microprocessors. Its function is to interface peripheral equipment to the microcomputer system bus. The data I/O bus of the 8255A are the lines marked D0-D7. Input and output instructions from the microprocessor change the states of the RD*, WR* and CS* lines (read, write and chip select respectively) which in turn control the 8255A data I/O bus and determine whether it will be used for input, output or whether it will be disabled (in a high-impedance state).

The CS* pin is the Chip Select for the 8255A. A CS* pin can be thought of as a master select pin because unless it is in its active state (low) the 8255A is inactive and its data I/O bus is in a high-impedance state and all of its control pins are ignored (except RESET). A CS* pin is common among microprocessor peripherals and memories because it allows many devices to use a common data bus by allowing the microprocessor and its support circuitry to control which device will use the data bus.

If the 8255A's CS* pin is low, it is selected and the RD* and WR* pins determine whether data will be read from or written to it, and the A0 and A1 pins (address bus pins) determine which of the 3 read registers and 4 write registers will be used. This is shown in the chart below.

PORT SELECT CHARACTERISTICS					
(READ FROM 8255A)					
A1	**A0**	**RD***	**WR***	**CS***	
0	0	0	1	0	Port A
0	1	0	1	0	Port B
1	0	0	1	0	Port C
1	1	0	1	0	(Illegal condition)
(WRITE TO 8255A)					
A1	**A0**	**RD***	**WR***	**CS***	
0	0	1	0	0	Port A
0	1	1	0	0	Port B
1	0	1	0	0	Port C
1	1	1	0	0	Control register

(DISABLE 8255A)					
A1	A0	RD*	WR*	CS*	
X	X	X	X	1	3-state
1	1	0	1	0	Illegal
X	X	1	1	0	3-state

There are three modes of operation that can be selected by the system software.

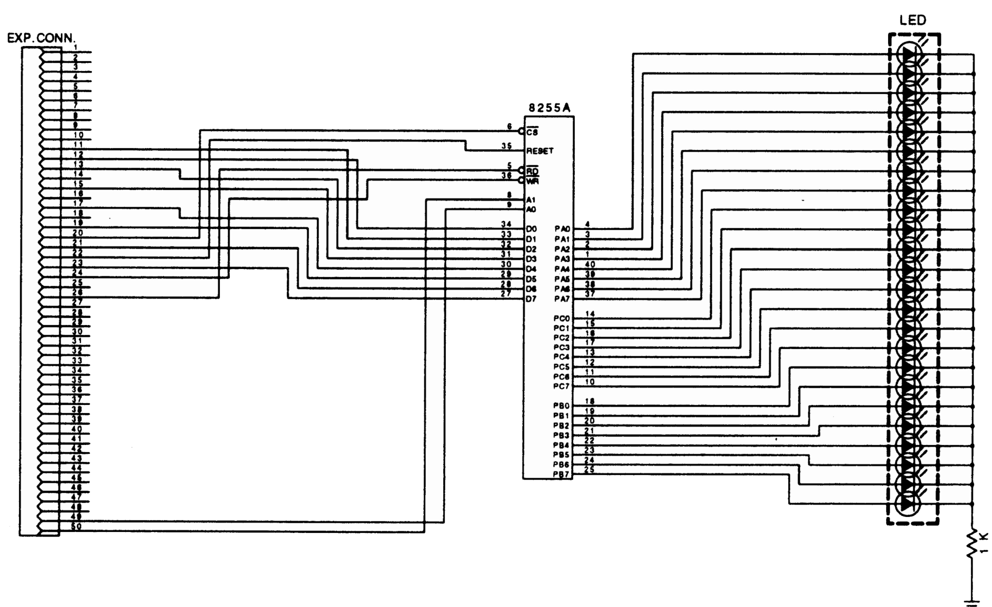
Mode 0 - Basic input/output
Mode 1 - Strobed Input/output
Mode 2 - Bi-Directional Bus

For this experiment we used mode 0. In this mode, the 8255A has three 8 bit I/O ports (ports A, B and C) which can be individually configured as inputs or outputs. Port C is unique in that it can be treated as two 4 bit ports which are programmed individually as inputs or outputs. When a "high" is seen at the 8255A's RESET pin, this clears all the internal registers, including the control register, and all ports are set to the input mode. In the circuit described below, the RESET pin is connected to the PRIMER reset circuit so the 8255A can be reset when the PRIMER reset button is pressed or when the PRIMER is powered up.

Circuit Description

Refer to the schematic. The 8255A adapts easily to the 8085 architecture since it was originally designed to be an 8080/8085 peripheral. The necessary control lines and busses are on the expansion connector CN1 and have the same labels as the 8255A pins, except for EXTIOCS*. The EXTIOCS* is a I/O chip select output that is decoded on-board which is connected to CS* of the 8255A. The I/O address range where EXTIOCS* is active is from 0C0H to 0FFH. Since we are only using address lines A0 and A1 addresses 0C0H to 0C3H can be used to select the 8255A registers and ports.

The pins of ports A, B and C will be connected to LED's which are in turn connected to a common current limiting resistor. Note that it is allowable to use a common resistor if only one LED is active at a time. If a program is written which turns on more than one at a time, the LED's will become dim and you could possibly burn out the resistor if its power rating is too low.



The Vcc and ground pins are not shown on the schematic. Ground will come from pin 27 of CN1 and go to pin 7 of the 8255A (note that all references to pin numbers in this application are based on a 40 pin DIP package pinout). The section of wire-wrap wire can be used to connect the Vcc (+5v) supply available on CN3 pin 21 or 22, to pin 26 of the 8255A. If it is desired to have more than one LED on at a time, you should power the circuit with a separate 5v supply and install (24) 1k ohm resistors between ground and each LED. You will also need to determine the maximum power dissipation of your particular 8255A to make sure the load applied doesn't damage it.

All connections to the PRIMER will be made by connecting one end of a 50 pin ribbon cable to the expansion connector and using jumper wires to connect the other end to the breadboard. To make the 50 pin ribbon cable, we need to orient the ribbon and the 50 pin connectors so that when the cable is assembled and plugged into the PRIMER, the female connector on the other end is pointing up. Most 50 pin female connectors have an arrow or mark indicating pin 1. Orient the connector so it will connect to pin 1 of the header when the ribbon is pointed away from the board. Similarly, some 50 wire ribbon cables have one edge wire that is marked in some way. If your cable is like this, the convention is to orient the cable so the marked wire is on the same side as pin 1 of the header. On the other end of the cable, the female connector should point up, with the female header mark for pin 1 on the same edge of the cable as the mark on the other female header. When the headers are properly oriented on the ribbon cable, they should be pressed into the cable wire with a vise. (Only apply enough pressure to close the protective back onto the header connector or it could be damaged). When the cable is made this way, pin 1 is easily found on the cable and it can be used as a reference to find the other pins needed for this application.

Program Execution

The program lights up 24 LED's in sequential order, one LED at a time. The sequence is: port A, port C, port B, repeat. The current port in the sequence starts with bit 0 high, and moves bit by bit to bit 7 then all its bits are cleared and the bit pattern is followed in the next port in sequence.

Refer to the assembly language listing below. The 8255A is put in mode 0, and Ports A, B, and C are programmed as outputs to drive the LED's. The carry flag is set and the accumulator is cleared, then the main loop is entered. The main loop has three loops nested within it: one for port A, C and B and they are executed in that order. Each of the nested loops perform the same function but for different ports. They rotate the carry bit through the accumulator and before each display there is a CALL to a delay routine to allow the previous output LED to be shown long enough to tell us where the bit is within the 24 port pins. When the carry bit has rotated out of the accumulator the loop falls through to the next nested loop. When all three nested loops are finished the program jumps back to the first nested loop.

The assembly language code is listed below:

```
PORTA    EQU     0C0H        ; 8255 PORT A
PORTB    EQU     0C1H        ; 8255 PORT B
PORTC    EQU     0C2H        ; 8255 PORT C
CONTRL   EQU     0C3H        ; CONTROL REG
DELAY    EQU     14H         ; SERVICE FOR READING KEYPAD
MOS      EQU     1000H       ; MOS CALL ADDRESS

         ORG     0FF01H
         MVI     A,80H       ; CONFIGURE MODE 0 WITH ALL PORTS OUTPUT
         OUT     CONTRL      ; WRITE TO CONTROL REG.

         MVI     A,0         ; START WITH ACC=0
         STC                 ; SET CY

SHPRTA:  CALL    SHFTDLY     ; SHIFT ACC WITH CY
         OUT     PORTA
         JNC     SHPRTA      ; LOOP TILL CY SET
```

```
SHPRTC:  CALL     SHFTDLY     ; SHIFT ACC WITH CY
         OUT      PORTC
         JNC      SHPRTC      ; LOOP TILL CY SET

SHPRTB:  CALL     SHFTDLY     ; SHIFT ACC WITH CY
         OUT      PORTB
         JNC      SHPRTB      ; LOOP TILL CY SET

         JMP      SHPRTA      ; DO PORT A AGAIN
;
; Rotate the Acc with the CY and delay if CY not set.
;
SHFTDLY: MVI      C,DELAY     ; SELECT THE DELAY SERVICE
         LXI      H,8000H     ; DELAY PERIOD
         CNC      MOS         ; DO A MOS SERVICE CALL IF NO CY
         RAL                  ; ROTATE LEFT THROUGH CY
         RET
```

Load the following machine language program into memory:

ADDRESS	**DATA**	**DESCRIPTION**	**ADDRESS**	**DATA**	**DESCRIPTION**
FF01	3E	MVI A,80	FF17	FF	
FF02	80		FF18	CD	CALL FF23
FF03	D3	OUT C3	FF19	23	
FF04	C3		FF1A	FF	
FF05	3E	MVI A,00	FF1B	D3	OUT C1
FF06	00		FF1C	C1	
FF07	37	STC	FF1D	D2	JNC FF18
FF08	CD	CALL FF23	FF1E	18	
FF09	23		FF1F	FF	
FF0A	FF		FF20	C3	JMP FF08
FF0B	D3	OUT C0	FF21	08	
FF0C	C0		FF22	FF	
FF0D	D2	JNC FF08	FF23	0E	MVI C,14
FF0E	08		FF24	14	
FF0F	FF		FF25	21	LXI H,8000
FF10	CD	CALL FF23	FF26	00	
FF11	23		FF27	80	
FF12	FF		FF28	D4	CNC 1000
FF13	D3	OUT C2	FF29	00	
FF14	C2		FF2A	10	
FF15	D2	JNC FF10	FF2B	17	RAL
FF16	10		FF2C	C9	RET

Application 11: Pulse Tone Dialer

Purpose

To construct a phone dialer.

Goals

1. Build and test an autodialer circuit.
2. Load a program that will initialize the DTMF chip and send a phone number stored in memory.

Materials

Qty.	Description
1	Primer Trainer
1	RJ-11C Phone Jack
1	CH1817 DAA Module
1	MT8889C Integrated DTMF Transceiver
1	TTL 7404
6	0.1 µF capacitors
1	22 µF capacitor
2	100 KΩ resistors
1	375 KΩ resistor
1	3.8 MHz crystal
1	10 KΩ resistor

Overview

This circuit is built around the MT8889C DTMF Transceiver. This chip has several internal registers that can be used for status, control, and data. Access to these registers is by way of pin 49 from the Primer bus expansion bus header to pin 9 on the MT8889C. The state of this pin controls which ports to write or read from. Port 0C0h is used for read/write access to the data register, while port 0C1h is used to write to the control registers or read the status register. The chip also has a status line that can be read from the I/O expansion bus, pin number 3. This status line can be polled to see if a valid DTMF tone has been transmitted or received.

The receiver can be ordered from Bell Industries (www.bellind.com), 1-800-289-2355; Jaco Electronics, 1-800-989-5226; or Sterling Electronics (www.sterling.com), 1-800-745-5500.

Also present in this circuit is a CH1817 DAA module. The purpose of this module is to provide an FCC approved interface to any phone system. This module connects directly to the telephone line on one side and to the transceiver on the other. It sets the line either on or off-hook and takes the DTMF signal generated by the transceiver and puts it onto the line. Access to this is through the I/O expansion bus, pin #2. This line is inverted so that sending a 0 will set the line to off-hook. Once set to off-hook the chip can then put the DTMF signals onto the telephone line. After all of the tones have been sent it should then be set back to on-hook to free up the line. More information on the internal functions is located in Table 1.

The DAA module can be ordered directly from Cermetek (www.cermetek.com), 1-800-882-6271.

Program Description

The program first sets the DAA module to off-hook and then initializes the MT8889C (the complete initialization sequence is found in Table 5). Once the MT8889C has been initialized, the software then sets the operating parameters of the chip. The software sets it to enable the tone output, send DTMF tones, interrupt enabled, and burst mode. A more complete description of the usage of the control registers and their bits are located in Tables 2 and 3. The program then waits for a key (0-4) on the Primer Trainer to be pressed. Once pressed it will jump to the address of the phone number that had been previously been entered in memory. Warning: the phone numbers that have been defined in this program should not be used to actually dial out. You should disconnect the phone line immediately after dialing in order to avoid annoying a person which might actually have one of these numbers and to avoid long-distance charges. The safest solution is to redefine these phone numbers to your own number or to known local numbers. This program uses a static algorithm and therefore each phone number string must be 12 bytes long and terminate with a 0FFh. A number that is less than 11 digits must be padded on the left side by 0FFh (modifying the code to take dynamic numbers is left as an challenge to the reader). Once the program jumps to the first memory location of the phone number it checks the condition of the transceiver to ensure that it is ready to accept data. When it is ready it sends the tone and waits for the transceiver to pass it off to the DAA module, which in turn puts it onto the phone line. In addition it also shows the digit being sent on display #2. The program then polls the status register on the transceiver to check if it is ready and once it is ready it will send the second tone. More information on the status register is located in Table 4 and a sample control routine is located in Table 6. This continues until the program reads an FFh and at that point it breaks out from the loop and resets the DAA module to on-hook so that the line is freed up for other devices. If a phone is hooked up in parallel, or if one uses a telephone line simulator one can hear the DTMF tones being sent from the DAA module down the line.

RS0	WR	RD	FUNCTION
0	0	1	Write to Transmit Data Register
0	1	0	Read from Receive Data Register
1	0	1	Write to Control Register
1	1	0	Read from Status Register

Table 1
Internal Register Functions

BIT	NAME	DESCRIPTION
B0	TOUT	Tone Output Control. A logic high enables the tone output; a logic low turns the tone output off. This bit controls all transmit tone functions.
B1	CP / DTMF	Call Progress or DTMF Mode Select. A logic high enables the receive call progress mode; a logic low enables DTMF mode.
B2	IRQ	Interrupt Enable. A logic high enables the interrupt function; a logic low de-activates the interrupt function.
B3	RSEL	Register Select. A logic high selects control register B for the next write cycle to the control register.

Table 2
Control Register A Description

BIT	NAME	DESCRIPTION
B0	BURST	Burst Mode Select.
B1	TEST	Test Mode Control. A logic high enables the test mode; a logic low de-activates the test mode.
B2	S / D	Single or Dual Tone Generation. A logic high selects the single tone output; a logic low selects the dual tone (DTMF) output.
B3	RSEL	Column or Row Tone Select. A logic high selects a column tone output; a logic low selects a row tone output.

Table 3
Control Register B Description

BIT	NAME	STATUS FLAG SET	STATUS FLAG CLEARED
B0	IRQ	Interrupt has occurred. Bit one (b1) or bit two (b2) is set.	Interrupt is inactive. Cleared after Status Register is read.
B1	TRANSMIT DATA REGISTER EMPTY	Transmitter is ready for new data	Cleared after Status Register is read or when in non-burst mode.
B2	RECEIVE DATA REGISTER FULL	Valid data is in the Receive Data Register	Cleared after Status Register is read.
B3	DELAYED STEERING	Set upon the valid detection of the absence of a DTMF signal.	Cleared upon the detection of a valid DTMF signal.

Table 4
Status Register Description

INITIALIZATION PROCEDURE

A software reset must be included at the beginning of all programs to initialize the control registers after a power up. The initialization procedure should be implemented 100 ms after power up.

			DATA			
Description:	**WR**	**RD**	**B0**	**B1**	**B2**	**B3**
1. Read Status Register	1	0	X	X	X	X
2. Write to Control Register	0	1	0	0	0	0
3. Write to Control Register	0	1	0	0	0	0
4. Write to Control Register	0	1	1	0	0	0
5. Write to Control Register	0	1	0	0	0	0
6. Read Status Register	1	0	X	X	X	X

Table 5

TYPICAL CONTROL SEQUENCE FOR BURST MODE APPLICATIONS					
Transmit DTMF tones of 50 ms burst/50 ms pause and Receive DTMF Tones.					
Sequence:		**DATA**			
	RS0	**B3**	**B2**	**B1**	**B0**
1. Write to Control Register A	1	1	1	0	1
2. Write to Control Register B	1	0	0	0	0
3. Write to Transmit Data Register (send a digit 7)	0	0	1	1	1
4. Wait for an Interrupt or Poll Status Register					
5. Read the Status Register	1	X	X	X	X
• If bit 1 is set, the Tx is ready for the next tone. Write to Transmit Register (send a five)	0	0	1	0	1
• If bit 2 is set, a DTMF tone has been received Read the Receive Data Register	0	X	X	X	X
• If both bits are set Read the Receive Data Register	0	X	X	X	X
Write to Transmit Data Register	0	0	1	0	1

Table 6

The assembly language code is listed below:

```
STARTADD     EQU   0FF01h
OFFHOOK      EQU   0FEh

             ORG   STARTADD    ; STARTING ADDRESS IN MEMORY
MAIN:        MVI   A,OFFHOOK   ; SET LEAST SIG BIT TO 0
             OUT   011h        ; SET DAA MODULE TO OFF-HOOK
             ; INITALIZATION SEQUENCE FOR MT8889C
             IN    0C1h        ; READ STATUS REGISTER
             MVI   A,00h       ; BYTES TO BE SENT TO CONTROL REGISTER
             OUT   0C1h        ; WRITE TO CONTROL REGISTER
             OUT   0C1h        ; WRITE AGAIN TO CONTROL REGISTER
             MVI   A,08h       ; SET LEFT BIT HIGH ON FIRST BYTE
             OUT   0C1h        ; WRITE TO CONTROL REGISTER
             MVI   A,00h       ; RESET THE BYTE
             OUT   0C1h        ; WRITE IT TO CONTROL REGISTER
             IN    0C1h        ; READ THE STAUS REGISTER
             ; END INITALIZATION SEQUENCE

             MVI   A,0Dh       ; SET MT8889C TO TONE OUT, DTMF, IRQ,
                               ; SELECT CONTROL REGISTER B
             OUT   0C1h        ; WRITE TO CONTROL REGISTER A
             MVI   A,00h       ; BURST MODE
             OUT   0C1h        ; WRITE TO CONTROL REGISTER B
             MVI   C,0Bh       ; SERVICE KEYIN
             CALL  1000h       ; CALL SERVICE
             MOV   H,L         ; MOVE THE KEYVALUE TO H
             MOV   D,L         ; MOVE THE KEYVALUE TO D

             ; MULTIPLY BY 12 (X =((Y*2*2)+ Y + Y)*2
             DAD   H           ; ADD H TO REGISTER PAIR HL (Y*2)
             DAD   H           ; ADD H TO REGISTER PAIR      HL (*2)
```

```
            DAD   D           ; ADD D TO REGISTER PAIR HL (+Y)
            DAD   D           ; ADD D TO REGISTER PAIR HL (+Y)
            DAD   H           ; ADD H TO REGISTER PAIR HL (*2)
            MOV   C,H         ; MOVE THE OFFSET TO C
            MVI   B,00h       ; INITALIZE THE LEFT BYTE OF BC REGISTER PAIR
            LXI   H,PHONE     ; GO TO FIRST ADDRESS OF PHONE NUMBER ARRAY
            DAD   B           ; ADD THE OFFSET VALUE IN BC TO HL

LOOP:       MOV   A,M         ; GET THE PHONE NUMBER DIGIT
            MOV   E,A         ; SEND IT TO REGISTER E FOR DISPLAY
            MVI   B,0FFh      ; TERMINATION VALUE
            MVI   C,012h      ; SERVICE 012h
            CMP   B           ; IS LAST VALUE THE TERMINATION VALUE
            JZ    EXIT        ; IF SO, DONE
            OUT   0C0h        ; ELSE WRITE DIGIT TO DTMF TRANSCEIVER
            MVI   d,00h       ; CLEAR LEFT DISPLAY BYTE
            MVI   H,0FFh      ; DELAY TIME
            CALL  1000h       ; CALL SERVICE 12h
            MVI   C,014h      ; SERVICE 14h
            CALL  1000h       ; CALL SERVICE 14h

READ:       IN    0C1h        ; READ THE STATUS REGISTER
            ANI   02h         ; AND VALUE WITH 02hMVI       D,02h ; STORE 02h
IN D
            CMP   D           ; CMP 02h WITH ANDed STATUS
            JZ    INCREMENT   ; IF = 0 READY FOR NEXT DIGIT
            JMP   READ        ; ELSE WAIT ANOTHER CYCLE FOR BUFFER TO FLUSH

INCREMENT:  INX   H           ; MOVE TO NEXT DIGIT
            JMP   LOOP        ; GO BACK AND DO IT ALL OVER AGAIN
EXIT:       MVI   A,0FFh      ; SET DAA MODULE TO ON-HOOK
            OUT   11h         ; WRITE TO OUTPUT PORT

            ; PHONE NUMBERS - ONE DIGIT PER BYTE, MUST HAVE 12 VALUES (11
DIGITS MAX
            ; AND ONE FFh VALUE TO SIGNIFY TERMINATION OF SEQUENCE
            ; **WARNING** the phone numbers that have been defined in this
            ; program should not be used to actually dial out. You should
            ; disconnect the phone line immediately after dialing in order to
            ; avoid annoying a person which might actually have one of these
            ; numbers and to avoid long-distance charges.

            ; 123-4567
PHONE       DB    1h,2h,3h,4h,5h,6h,7h,0FFh,0FFh,0FFh,0FFh,0FFh

            ; 987-6543
            DB    9h,8h,7h,6h,5h,4h,3h,0FFh,0FFh,0FFh,0FFh,0FFh

            ; 456-7890
            DB    4h,5h,6h,7h,8h,9h,0h,0FFh,0FFh,0FFh,0FFh,0FFh

            ; 1-800-483-3737 (GTE long distance service -as of 12/27/99-)
            DB    1h,8h,0h,0h,4h,8h,3h,3h,7h,3h,7h,0FFh
            END
```

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION	
FF01	3E	MVI	A,0FEh
FF02	FE		
FF03	D3	OUT	011h
FF04	11		
FF05	DB	IN	0C1h
FF06	C1		
FF07	3E	MVI	A,00h
FF08	00		
FF09	D3	OUT	0C1h
FF0A	C1		
FF0B	D3	OUT	0C1h
FF0C	C1		
FF0D	3E	MVI	A,08h
FF0E	08		
FF0F	D3	OUT	0C1h
FF10	C1		
FF11	3E	MV	A,00h
FF12	00		
FF13	D3	OUT	0C1h
FF14	C1		
FF15	DB	IN	0C1h
FF16	C1		
FF17	3E	MVI	A,0Dh
FF18	0D		
FF19	D3	OUT	0C1h
FF1A	C1		
FF1B	3E	MVI	A,00h
FF1C	00		
FF1D	D3	OUT	0C1h
FF1E	C1		
FF1F	0E	MVI	C,0Bh
FF20	0B		
FF21	CD	CALL	1000h
FF22	00		
FF23	10		
FF24	65	MOV	H,L
FF25	55	MOV	D,L
FF26	29	DAD	H
FF27	29	DAD	H
FF28	19	DAD	D
FF29	19	DAD	D
FF2A	29	DAD	H
FF2B	4C	MOV	C,H
FF2C	06	MVI	B,00h
FF2D	00		
FF2E	21	LXI	H,PHONE
FF2F	5D		
FF30	FF		
FF31	09	DAD	B
FF32	7E	MOV	A,M
FF33	5F	MOV	E,A
FF34	06	MVI	B,0FFh
FF35	FF		
FF36	0E	MVI	C,012h
FF37	12		
FF38	B8	CMP	B
FF39	CA	JZ	EXIT
FF3A	5B		
FF3B	FF		
FF3C	D3	OUT	0C0h
FF3D	C0		
FF3E	16	MVI	D,00h
FF3F	00		
FF40	26	MVI	H,0FFh
FF41	FF		
FF42	CD	CALL	1000h
FF43	00		
FF44	10		
FF45	0E	MVI	C,014h
FF46	14		
FF47	CD	CALL	1000h
FF48	00		
FF49	10		
FF4A	DB	IN	0C1h
FF4B	C1		
FF4C	E6	ANI	02h
FF4D	02		
FF4E	16	MVI	D,02h
FF4F	02		
FF50	BA	CMP	D
FF51	CA	JZ	INCREMENT
FF52	57		
FF53	FF		
FF54	C3	JMP	READ
FF55	4A		
FF56	FF		
FF57	23	INX	H
FF58	C3	JMP	LOOP
FF59	32		
FF5A	FF		
FF5B	3E	MVI	A,0FFh
FF5C	FF		
FF5D	D3	OUT	11h
FF5E	11		
FF5F	01	DB	1h,2h,3h,4h, 5h,6h,7h,0FFh, 0FFh,0FFh, 0FFh,0FFh
FF60	02		
FF61	03		
FF62	04		
FF63	05		
FF64	06		
FF65	07		
FF66	FF		
FF67	FF		

Continued on next page…

ADDRESS	DATA	DESCRIPTION	
FF68	FF		
FF69	FF		
FF6A	FF		
FF6B	09	DB	9h,8h,7h,6h, 5h,4h,3h,0FFh, 0FFh,0FFh, 0FFh,0FFh
FF6C	08		
FF6D	07		
FF6E	06		
FF6F	05		
FF70	04		
FF71	03		
FF72	FF		
FF73	FF		
FF74	FF		
FF75	FF		
FF76	FF		
FF77	04	DB	4h,5h,6h, 7h,8h,9h,0h, 0FFh,0FFh, 0FFh,0FFh, 0FFh
FF78	05		
FF79	06		

ADDRESS	DATA	DESCRIPTION	
FF7A	07		
FF7B	08		
FF7C	09		
FF7D	00		
FF7E	FF		
FF7F	FF		
FF80	FF		
FF81	FF		
FF82	FF		
FF83	01	DB	1h,8h,0h, 0h,4h,8h,3h, 3h,7h,3h,7h, 0FFh
FF84	08		
FF85	00		
FF86	00		
FF87	04		
FF88	08		
FF89	03		
FF8A	03		
FF8B	07		
FF8C	03		
FF8D	07		
FF8E	FF		

Application 12: Pulse Tone Receiver

Purpose

To construct a phone receiver

Goals

1. Build and test a receiving circuit.
2. Load a program that will initialize the DTMF chip and perform actions on two relays from remote DTMF signals.

Materials

Qty.	Description
1	Primer Trainer
1	RJ-11C Phone Jack
1	CH1817 DAA Module
1	MT8889C Integrated DTMF Transceiver
1	TTL 7404
6	0.1 µF capacitors
1	22 µF capacitor
2	100 KΩ resistors
1	375 KΩ resistor
1	3.8 MHz crystal
1	10 KΩ resistor

Overview

This application has a variety of uses. It can be used in any situation where ones requires the ability to remotely access a device through a telephone line. For example, one could use this device to remotely turn a set of lights on or off or turn a series of industrial relays on or off. It is designed to run in a continuous mode and represents how an embedded application should execute.

This circuit is built around the MT8889C DTMF Transceiver. This chip has several internal registers that can be used for status, control, and data. Access to these registers is by way of pin 49 from the Primer bus expansion bus header to pin 9 on the MT8889C. The state of this pin controls which ports to write or read from. Port 0C0h is used for read/write access to the data register, while port 0C1h is used to write to the control registers or read the status register. The chip also has a status line that can be read from the I/O expansion bus, pin number 3. This status line can be polled to see if a valid DTMF tone has been transmitted or received.

The receiver can be ordered from Bell Industries (www.bellind.com), 1-800-289-2355; Jaco Electronics, 1-800-989-5226; or Sterling Electronics (www.sterling.com), 1-800-745-5500.

Also present in this circuit is a CH1817 DAA module. The purpose of this module is to provide an FCC approved interface to any phone system. This module connects directly to the telephone line on one side and to the transceiver on the other. It sets the line either on or off hook and takes the DTMF signal generated by the transceiver and puts it onto the line. Access to this is through the I/O expansion bus, pin #2. This line is inverted so that sending a 0 will set the line to off hook. Once set to off hook the chip can then put the DTMF signals onto the telephone line. After all of the tones have been sent it should then be set back to on hook to free up the line. More information on the internal functions is located in Table 1.

The DAA module can be ordered directly from Cermetek (www.cermetek.com), 1-800-882-6271.

Program Description

The program can control two relays which are signified by two LEDs turning off and on. Number one and number two on the remote phone keypad will turn LD4 and LD5 on, respectively. Number three and four will turn LD4 and LD5 respectively as well. The * on the remote phone keypad will turn off both LEDs. Once the LEDs have been set to the desired state one can reset the phone line and cause the Trainer to hang up the line by pressing the # key. One can then dial back into the Trainer after any specified amount of time to change the states of the LEDs. The program does this by running in a continual loop.

The program first initializes the MT8889C (the complete initialization sequence can be located in Table 5) . Once the MT8889C has been initialized the software then sets the operating parameters of the chip. The software sets it to enable the tone output, send DTMF tones, interrupt enabled, and burst mode. A more complete description of the usage of the control registers and their bits are located in Table 2 and 3. The program waits for the DAA module to recognizes a ring and once it does so it will then set the DAA module to off hook and send three tones to the remote phone to signify that it has connected. Once connected the program listens for a 1,2,3,4,*, or # and performs the actions described above. When one is done with modifying the relays they must tell the device to hang up if they wish to free the line up. The DAA module will not reset itself.

RS0	WR	RD	FUNCTION
0	0	1	Write to Transmit Data Register
0	1	0	Read from Receive Data Register
1	0	1	Write to Control Register
1	1	0	Read from Status Register

TABLE 1
Internal Register Functions

BIT	NAME	DESCRIPTION
B0	TOUT	Tone Output Control. A logic high enables the tone output; a logic low turns the tone output off. This bit controls all transmit tone functions.
B1	CP / DTMF	Call Progress or DTMF Mode Select. A logic high enables the receive call progress mode; a logic low enables DTMF mode.
B2	IRQ	Interrupt Enable. A logic high enables the interrupt function; a logic low de-activates the interrupt function.
B3	RSEL	Register Select. A logic high selects control register B for the next write cycle to the control register.

TABLE 2
Control Register A Description

BIT	NAME	DESCRIPTION
B0	BURST	Burst Mode Select.
B1	TEST	Test Mode Control. A logic high enables the test mode; a logic low de-activates the test mode.
B2	S D	Single or Dual Tone Generation. A logic high selects the single tone output; a logic low selects the dual tone(DTMF) output.
B3	RSEL	Column or Row Tone Select. A logic high selects a column tone output; a logic low selects a row tone output.

TABLE 3
Control Register B Description

BIT	NAME	STATUS FLAG SET	STATUS FLAG CLEARED
B0	IRQ	Interrupt has occurred. Bit one (b1) or bit two (b2) is set.	Interrupt is inactive. Cleared after Status Register is read.
B1	TRANSMIT DATA REGISTER EMPTY	Transmitter is ready for new data	Cleared after Status Register is read or when in non-burst mode.
B2	RECEIVE DATA REGISTER FULL	Valid data is in the Receive Data Register	Cleared after Status Register is read.
B3	DELAYED STEERING	Set upon the valid detection of the absence of a DTMF signal.	Cleared upon the detection of a valid DTMF signal.

TABLE 4
Status Register Description

INITIALIZATION PROCEDURE

A software reset must be included at the beginning of all programs to initialize the control registers after a power up. The initialization procedure should be implemented 100 ms after power up.

			DATA			
Description:	**WR**	**RD**	**B0**	**B1**	**B2**	**B3**
1. Read Status Register	1	0	X	X	X	X
2. Write to Control Register	0	1	0	0	0	0
3. Write to Control Register	0	1	0	0	0	0
4. Write to Control Register	0	1	1	0	0	0
5. Write to Control Register	0	1	0	0	0	0
6. Read Status Register	1	0	X	X	X	X

TABLE 5

TYPICAL CONTROL SEQUENCE FOR BURST MODE APPLICATIONS					
Transmit DTMF tones of 50 ms burst/50 ms pause and Receive DTMF Tones.					
Sequence:		**DATA**			
	RS0	**B3**	**B2**	**B1**	**B0**
1. Write to Control Register A	1	1	1	0	1
2. Write to Control Register B	1	0	0	0	0
3. Write to Transmit Data Register (send a digit 7)	0	0	1	1	1
4. Wait for an Interrupt or Poll Status Register					
5. Read the Status Register	1	X	X	X	X
• If bit 1 is set, the Tx is ready for the next tone. Write to Transmit Register (send a five)	0	0	1	0	1
• If bit 2 is set, a DTMF tone has been received Read the Receive Data Register	0	X	X	X	X
• If both bits are set Read the Receive Data Register	0	X	X	X	X
Write to Transmit Data Register	0	0	1	0	1

TABLE 6

The assembly language code is listed below:

```
            ORG   0FF01h

MAIN:       IN    0C1h      ;INITIALIZATION STRING
            MVI   A,00h
            OUT   0C1h
            OUT   0C1h
            MVI   A,08h
            OUT   0C1h
            MVI   A,00h
            OUT   0C1h
            IN    0C1h

            MVI   A,0Dh     ;TRANSCEIVER MODE SETUP
            OUT   0C1h
            MVI   A,00h
            OUT   0C1h
            MVI   C,14h     ;SERVICE ROUTINE SETUP
            MVI   H,06Fh    ;WAIT STATE
            MVI   A,0FEh
            STA   LED
ONHOOK:     MVI   E,01H     ;COUNTER

TONE:       MVI   B,0FEh    ;WAIT FOR A TONE
            IN    12h
            SUB   B
            JZ    PHONE     ;IF A TONE GO TO PHONE
            JMP   TONE      ;GO AND WAIT FOR IT AGAIN

PHONE:      LDA   LED       ;SET DAA MODULE OFFHOOK
            ANI   0FEh
            OUT   11h
            CALL  1000h     ;CALL THE WAIT STATE
            MVI   A,04h
DIGIT:      OUT   0C0h      ;SEND A 4
LOOP1:      IN    0C1h      ;READ THE TRANSCEIVER STATUS REGISTER
            ANI   02h       ;CHECK TO SEE IF IT IS READY FOR NEXT BIT
            MVI   D,02h
            CMP   D         ;CHECK THE STATUS BIT FOR SENDING
            JZ    NEXT
            JMP   LOOP1

NEXT:       INR   E
            MVI   A,03h
            CMP   E
            CALL  1000h
            JNZ   DIGIT

READIN:     OUT   0C0h      ;SEND THE LAST DIGIT
            IN    0C1h      ; READ THE STATUS REGISTER
            CALL  1000h     ;CALL A WAIT STATE

LOOP2:      IN    0C1h
            MVI   E,0C5h    ;CHECK THE VALID DTMF BIT
            ORA   E
            CMP   E
```

```
              JNZ   LOOP2

INPUT:        CALL  1000h
              IN    0C0h      ;GET THE INPUT BIT
              CALL  1000h
              CPI   0CCh
              JZ    EXIT
              CPI   0CBh
              MVI   E,0FEh
              JZ    ALLOFF
              CPI   0C1h
              MVI   E,0EEh
              JZ    RELAYON
              CPI   0C2h
              MVI   E,0DEh
              JZ    RELAYON
              CPI   0C3h
              MVI   E,0DEh
              JZ    RELAYOFF
              CPI   0C4h
              MVI   E,0EEh
              JZ    RELAYOFF
LOOP3:        JMP   LOOP2

RELAYOFF:     LDA   LED
              ORA   E
              JMP SOUND

RELAYON:      LDA   LED
              ANA   E
SOUND:        STA   LED
              OUT   11h
              MVI   A,07h
              OUT   0C0h
              CALL  1000h
              OUT   0C0h
              JMP   LOOP2

ALLOFF:       LDA   LED
              ORA   E
              JMP   SOUND

EXIT:         LDA   LED
              ORI   03h
              OUT   11h
              JMP   ONHOOK
LED:          DS    1
              END
```

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION	
FF01	DB	IN	0C1h
FF02	C1		
FF03	3E	MVI	A,00h
FF04	00		
FF05	D3	OUT	0C1h
FF06	C1		
FF07	D3	OUT	0C1h
FF08	C1		
FF09	3E	MVI	A,08h
FF0A	08		
FF0B	D3	OUT	0C1h
FF0C	C1		
FF0D	3E	MVI	A,00h
FF0E	00		
FF0F	D3	OUT	0C1h
FF10	C1		
FF11	DB	IN	0C1h
FF12	C1		
FF13	3E	MVI	A,0Dh
FF14	0D		
FF15	D3	OUT	0C1h
FF16	C1		
FF17	3E	MVI	A,00h
FF18	00		
FF19	D3	OUT	0C1h
FF1A	C1		
FF1B	0E	MVI	C,14h
FF1C	14		
FF1D	26	MVI	H,06Fh
FF1E	6F		
FF1F	3E	MVI	A,0FEh
FF20	FE		
FF21	32	STA	LED
FF22	C6		
FF23	FF		
FF24	1E	MVI	E,01H
FF25	01		
FF26	06	MVI	B,0FEh
FF27	FE		
FF28	DB	IN	12h
FF29	12		
FF2A	90	SUB	B
FF2B	CA	JZ	PHONE
FF2C	31		
FF2D	FF		
FF2E	C3	JMP	TONE
FF2F	26		
FF30	FF		
FF31	3A	LDA	LED
FF32	C6		
FF33	FF		
FF34	E6	ANI	0FEh
FF35	FE		
FF36	D3	OUT	11h

ADDRESS	DATA	DESCRIPTION	
FF37	11		
FF38	CD	CALL	1000h
FF39	00		
FF3A	10		
FF3B	3E		
FF3C	04	MVI	A,04h
FF3D	D3	OUT	0C0h
FF3E	C0		
FF3F	DB	IN	0C1h
FF40	C1		
FF41	E6	ANI	02h
FF42	02		
FF43	16	MVI	D,02h
FF44	02		
FF45	BA	CMP	D
FF46	CA	JZ	NEXT
FF47	4C		
FF48	FF		
FF49	C3	JMP	LOOP1
FF4A	3F		
FF4B	FF		
FF4C	1C	INR	E
FF4D	3E	MVI	A,03h
FF4E	03		
FF4F	BB	CMP	E
FF50	CD	CALL	1000h
FF51	00		
FF52	10		
FF53	C2	JNZ	DIGIT
FF54	3D		
FF55	FF		
FF56	D3	OUT	0C0h
FF57	C0		
FF58	DB	IN	0C1h
FF59	C1		
FF5A	CD	CALL	1000h
FF5B	00		
FF5C	10		
FF5D	DB	IN	0C1h
FF5E	C1		
FF5F	1E	MVI	E,0C5h
FF60	C5		
FF61	B3	ORA	E
FF62	BB	CMP	E
FF63	C2	JNZ	LOOP2
FF64	5D		
FF65	FF		
FF66	CD	CALL	1000h
FF67	00		
FF68	10		
FF69	DB	IN	0C0h
FF6A	C0		

Continued on next page…

ADDRESS	DATA	DESCRIPTION
FF6B	CD	CALL 1000h
FF6C	00	
FF6D	10	
FF6E	FE	CPI 0CCh
FF6F	CC	
FF70	CA	JZ EXIT
FF71	BC	
FF72	FF	
FF73	FE	CPI 0CBh
FF74	CB	
FF75	1E	MVI E,0FEh
FF76	FE	
FF77	CA	JZ ALLOFF
FF78	B5	
FF79	FF	
FF7A	FE	CPI 0C1h
FF7B	C1	
FF7C	1E	MVI E,0EEh
FF7D	EE	
FF7E	CA	JZ RELAYON
FF7F	A0	
FF80	FF	
FF81	FE	CPI 0C2h
FF82	C2	
FF83	1E	MVI E,0DEh
FF84	DE	
FF85	CA	JZ RELAYON
FF86	A0	
FF87	FF	
FF88	FE	CPI 0C3h
FF89	C3	
FF8A	1E	MVI E,0DEh
FF8B	DE	
FF8C	CA	JZ RELAYOFF
FF8D	99	
FF8E	FF	
FF8F	FE	CPI 0C4h
FF90	C4	
FF91	1E	MVI E,0EEh
FF92	EE	
FF93	CA	JZ RELAYOFF
FF94	99	
FF95	FF	
FF96	C3	JMP LOOP2
FF97	5D	
FF98	FF	

ADDRESS	DATA	DESCRIPTION
FF99	3A	LDA LED
FF9A	C6	
FF9B	FF	
FF9C	B3	ORA E
FF9D	C3	JMP SOUND
FF9E	A4	
FF9F	FF	
FFA0	3A	LDA LED
FFA1	C6	
FFA2	FF	
FFA3	A3	ANA E
FFA4	32	STA LED
FFA5	C6	
FFA6	FF	
FFA7	D3	OUT 11h
FFA8	11	
FFA9	3E	MVI A,07h
FFAA	07	
FFAB	D3	OUT 0C0h
FFAC	C0	
FFAD	CD	CALL 1000h
FFAE	00	
FFAF	10	
FFB0	D3	OUT 0C0h
FFB1	C0	
FFB2	C3	JMP LOOP2
FFB3	5D	
FFB4	FF	
FFB5	3A	LDA LED
FFB6	C6	
FFB7	FF	
FFB8	B3	ORA E
FFB9	C3	JMP SOUND
FFBA	A4	
FFBB	FF	
FFBC	3A	LDA LED
FFBD	C6	
FFBE	FF	
FFBF	F6	ORI 03h
FFC0	03	
FFC1	D3	OUT 11h
FFC2	11	
FFC3	C3	JMP ONHOOK
FFC4	24	
FFC5	FF	

Application 13: Reaction Tester

How fast we can react could mean the difference between getting into an auto accident and arriving at our destination on time. Or the difference between scoring the goal and winning the game or going home in defeat. Each of us has a different ability to react based upon our reflexes. Our reactions are affected by a number of factors such as, sleep, exhaustion, inebriation, etc. Studies also indicate that our reaction capability can be improved through training. The Reaction Tester can be used to monitor a person’s reaction time in order to determine at what level of performance the person is operating.

To use the Reaction Tester, the program listed below must be downloaded into the PRIMER. This can be accomplished by entering the machine code program in by hand or by assembling the program and downloading it to the PRIMER (Upgrade Option Required). Once the program is present in the PRIMER’s memory (be sure to double check the memory contents against the program), the Reaction Tester program is ready to run. To run the program, simply press the Reset button followed by pressing the Func then Run keys.

To use the Reaction Tester program the user simply pushes one of the keys on the numeric keypad. At this time the seven segment LED displays will read “9999” indicating to the user that it is waiting for a key press to start the reaction test. When a key is pressed the display will read 00.00. At some random period of time, approximately 1 to 10 seconds, after which the discrete LEDs will light and the speaker will sound. This indicates that the user is to press a key. Upon the user's key press the speaker sound will stop and the reaction time will be displayed on the LEDs. To initiate another reaction test the user simply presses a key and the reaction tester resets and 9999 is again displayed on the LEDs.

The Reaction Tester uses a truly random generator in order to prevent the user from anticipating the start. The program also checks for false starts, where the user jumps the gun and presses the keypad before allowed to do so. This program does not require any additional parts, everything required for the Application is included on a standard PRIMER. The user could however modify the program to use an external switch and light to enhance the Reaction Tester.

The Timer Equate used in the program is used to calibrate the Reaction Timer. This value can be modified in order to provide more accurate reaction timing. To calibrate the Reaction Timer use a stopwatch in conjunction with the Reaction Timer. Start the Reaction Timer at the same time you start the stopwatch. When the stopwatch reaches five seconds stop the Reaction Timer. If the Reaction Timer is greater than five seconds raise the Timer value. If the Reaction Timer is lower than five seconds lower the Timer value.

This Application makes extensive use of the MOS Services. The program uses six different service calls and accesses these six Services a total of twelve times. The MOS Services allow the user to easily make use of advanced, EMAC supplied, software modules in their programs (for more information on Services consult the Self Instruction Manual).

The assembly language code is listed below:

```
        ;*********************************************************
        ; Reaction Tester
        ; Copyright EMAC, Inc. For use with the Primer Trainer only.
        ;
        ; This Program tests the users reaction time. It uses a
        ; randomizer so the user can not anticipate the start.
        ; It also checks for false starts where the user jumps
        ; the gun.
        ;
        ; The program first displays 9999 on the display indicating
        ; that it is waiting for a key press to start the reaction
        ; test. When a key is pressed the display will read 00.00.
        ; At some random period of time, approximately 1 to 10
        ; seconds, after which the discrete LEDs will light and the
        ; speaker will sound. This indicates that the user is to
        ; press a key. Upon the user's key press the speaker sound
        ; will stop and the reaction time will be displayed on the
        ; LEDs. To initiate another reaction test the user simply
        ; presses a key and the reaction tester resets and 9999 is
        ; again displayed on the LEDs.
        ;
        ;*********************************************************

        = 0340   TIMER       EQU   0340h ; timing calibration constant
        = 1000   MOS         EQU   1000h ; MOS Service vector address

        = 000C   SERV0c      EQU   0ch   ; Discrete LED Service
        = 0010   SERV10      EQU   10h   ; Pitch Speaker Service
        = 0013   SERV13      EQU   13h   ; 7 Segment LED Service
        = 0014   SERV14      EQU   14h   ; Delay Service
        = 0016   SERV16      EQU   16h   ; Keypad Service
        = 0021   SERV21      EQU   21h   ; Decimal Point Service

        ORG      0FF01h
START:  MVI      C,SERV13    ; display 9999 on the LEDs-
        LXI      D,9999      ; this indicates ready to start
        CALL     MOS

        ; Get Random Number between 1 & 9
        MVI      C,SERV16
WAIT:   MVI      B,10        ; decrement random number
WAIT1:  DCR      B
        JZ       wait
        CALL     MOS         ; wait for a key press
        DCR      H
        JNZ      WAIT1

        MVI      C,SERV13    ; display 0000 on the LEDs
        LXI      D,0000h
        CALL     MOS
        MVI      C,SERV21    ; turn on the decimal point
        MVI      D,10h
        CALL     MOS
        MVI      C,SERV0c    ; turn off the discrete LEDs
        MVI      E,00h
```

```
         CALL     MOS

         ; Start Next Reaction Check at Random Interval
         INR      B
         LXI      H,0ffffh
         MVI      C,SERV14     ; lengthen delay by calling the-
DELAY:   CALL     MOS          ; delay service random # times
         DCR      B
         JNZ      DELAY



         MVI      C,SERV16     ; check for key press
         CALL     MOS
         DCR      H            ; if key press detected then-
         JZ       START        ; false start, start over

         MVI      C,SERV0c     ; turn on discrete LEDs
         MVI      E,0ffh
         CALL     MOS
         MVI      C,SERV10     ; turn on speaker output
         LXI      D,0c00h
         CALL     MOS

         LXI      D,0000h      ; check for reaction key press
         MVI      C,SERV16
LOOP:    LXI      H,TIMER      ; determine reaction time
LOOP1:   DCX      H
         MOV      A,H
         ORA      L
         JNZ      LOOP1
         INX      D
         CALL     MOS
         DCR      H
         JNZ      LOOP

         MVI      C,SERV13     ; output the current reaction-
         CALL     MOS          ; time to the 7 segment LEDs
         MVI      C,SERV21     ; add decimal point
         MVI      D,10h
         CALL     MOS
         MVI      C,SERV10     ; turn off speaker output
         LXI      D,0000h
CALL     MOS

         MVI      C,SERV16     ; check for key press
AGAIN:   CALL     MOS
         DCR      H
         JNZ      AGAIN        ; if key press then-
         JMP      START        ; start a new reaction test

         END
```

Load the following machine language program into memory:

ADDRESS	DATA	DESCRIPTION	
FF01	0E	MVI	C,13
FF02	13		
FF03	11	LXI	D,270F
FF04	0F		
FF05	27		
FF06	CD	CALL	1000
FF07	00		
FF08	10		
FF09	0E	MVI	C,16
FF0A	16		
FF0B	06	MVI	B,0A
FF0C	0A		
FF0D	05	DCR	B
FF0E	CA	JNZ	FF0B
FF0F	0B		
FF10	FF		
FF11	CD	CALL	1000
FF12	00		
FF13	10		
FF14	25	DCR	H
FF15	C2	JNZ	FF0D
FF16	0D		
FF17	FF		
FF18	0E	MVI	C,13
FF19	13		
FF1A	11	LXI	D,0000
FF1B	00		
FF1C	00		
FF1D	CD	CALL	1000
FF1E	00		
FF1F	10		
FF20	0E	MVI	C,21
FF21	21		
FF22	16	MVI	D,10
FF23	10		
FF24	CD	CALL	1000
FF25	00		
FF26	10		
FF27	0E	MVI	C,0C
FF28	OC		
FF29	1E	MVI	E,00
FF2A	00		
FF2B	CD	CALL	1000
FF2C	00		
FF2D	10		
FF2E	04	INR	B
FF2F	21	LXI	H,FFFF
FF30	FF		
FF31	FF		
FF32	0E	MVI	C,14
FF33	14		
FF34	CD	CALL	1000
FF35	00		
FF36	10		

ADDRESS	DATA	DESCRIPTION	
FF37	05	DCR	B
FF38	C2	JNZ	FF34
FF39	34		
FF3A	FF		
FF3B	0E	MVI	C,16
FF3C	16		
FF3D	CD	CALL	1000
FF3E	00		
FF3F	10		
FF40	25	DCR	H
FF41	CA	JZ	FF01
FF42	01		
FF43	FF		
FF44	0E	MVI	C,0C
FF45	0C		
FF46	1E	MVI	E,FF
FF47	FF		
FF48	CD	CALL	1000
FF49	00		
FF4A	10		
FF4B	0E	MVI	C,10
FF4C	10		
FF4D	11	LXI	D,0C00
FF4E	00		
FF4F	0C		
FF50	CD	CALL	1000
FF51	00		
FF52	10		
FF53	11	LXI	D,0000
FF54	00		
FF55	00		
FF56	0E	MVI	C,16
FF57	16		
FF58	21	LXI	H,0340
FF59	40		
FF5A	03		
FF5B	2B	DCR	H
FF5C	7C	MOV	A,H
FF5D	B5	ORA	L
FF5E	C2	FF5B	
FF5F	5B		
FF60	FF		
FF61	13	INX	D
FF62	CD	CALL	1000
FF63	00		
FF64	10		
FF65	25	DCR	H
FF66	C2	JNZ	FF58
FF67	58		
FF68	FF		
FF69	0E	MVI	C,13
FF6A	13		

Continued on next page…

ADDRESS	DATA	DESCRIPTION	
FF6B	CD	CALL	1000
FF6C	00		
FF6D	10		
FF6E	0E	MVI	C,21
FF6F	21		
FF70	16	MVI	D,10
FF71	10		
FF72	CD	CALL	1000
FF73	00		
FF74	10		
FF75	0E	MVI	C,10
FF76	10		
FF77	11	LXI	D,0000
FF78	00		
FF79	00		

ADDRESS	DATA	DESCRIPTION	
FF7A	CD	CALL	1000
FF7B	00		
FF7C	00		
FF7D	0E	MVI	C,16
FF7E	16		
FF7F	CD	CALL	1000
FF80	00		
FF81	10		
FF82	25	DCR	H
FF83	C2	JNZ	FF7F
FF84	7F		
FF85	FF		
FF86	C3	JMP	FF01
FF87	01		
FF88	FF		