VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI

MICROCONTROLLER LAB MANUAL

(18ECL47)

(As per Visvesvaraya Technological University Syllabus)Compiled By:

Prof. Sameera P
Assistant Professor, Dept. of ECE

Prof. Amulya D Raj
Assistant Professor, Dept. of ECE

Prof. Vaishnavi Pataki
Assistant Professor, Dept. of ECE

Department of ECE

Atria Institute of Technology

2020-21

CONTENTS

SYLLABUS

I. PROGRAMMING

1. Data Transfer - Block move, Exchange, Sorting, Finding largest element in an array
2. Arithmetic Instructions - Addition/subtraction, multiplication and division, square, Cube – (16 bits Arithmetic operations – bit addressable)
3. Counters
4. Boolean & Logical Instructions (Bit manipulations)
5. Conditional CALL & RETURN
6. Code conversion: BCD – ASCII; ASCII – Decimal; Decimal - ASCII; HEX - Decimal and Decimal - HEX
7. Programs to generate delay, Programs using serial port and on-Chip timer / counter

II. INTERFACING:

8. Write C programs to interface 8051 chip to Interfacing modules to develop single chip solutions
9. Simple Calculator using 6 digit seven segment display and Hex Keyboard interface to 8051
10. Alphanumeric LCD panel and Hex keypad input interface to 8051
11. External ADC and Temperature control interface to 8051
12. Generate different waveforms Sine, Square, Triangular, Ramp etc. using DAC interface to 8051; change the frequency and amplitude
13. Stepper and DC motor control interface to 8051
14. Elevator interface to 8051

PART I
PROGRAMMING

EXPERIMENT NO.1

DATA TRANSFER. - BLOCK MOVE, EXCHANGE, SORTING, FINDING LARGEST ELEMENT IN AN ARRAY

1.1 DATA TRANSFER

OBJECTIVE:
TO TRANSFER A BLOCK OF DATA BYTES FROM SOURCE MEMORY TO DESTINATION MEMORY USING 8051.

```
PROGRAM:
MOV R0,#50H       // Initialize the source memory pointer
MOV R1,#60H       // Initialize the destination memory pointer
MOV R2, #05H      // Initialize Iteration counter
BACK: MOV A,@R0   // Get the data from source memory pointer
MOV @R1,A         // Store the data into destination memory pointer
INC R0            // Increment the source memory pointer
INC R1            // Increment the destination memory pointer
DJNZ R2, BACK     // Decrement iteration count and if it
// is not zero,  go to relative Address and
// repeat the same process until count become
// zero.
END
MEMORY WINDOW:
Before execution:
D:0x50H:  22     AB     3D     44     55     00
D:0X60H:  00     00     00     00     00     00
After execution:
D:0x50H:  22     AB     3D     44     55     00
D:0X60H:  22     AB     3D     44     55     00
```

1. 2 BLOCK EXCHANGE

OBJECTIVE:
TO EXCHANGE TWO BLOCKS OF DATA BYTES USING 8051

```
PROGRAM:
MOV R0,#50H       // Initialize the source memory pointer
MOV R1,#60H       // Initialize the destination memory pointer
MOV R2,#05H       // Initialize  Iteration counter
BACK: MOV A,@R0   // Get the data from source memory pointer and Load
                  // into Accumulator
XCH A,@R1         // Exchange data between Accumulator and
                  // destination memory pointer
MOV @R0,A         // Store the data into source memory pointer
INC R0            // Increment the source memory pointer
INC R1            // Increment the destination memory pointer
DJNZ R2, BACK     /* Decrement iteration count and if it is not zero,
                  go to relative Address and repeat the same process
                  until count become zero*/
END
MEMORY WINDOW:
Before execution:
D:0x50H:  01      02      03      04     05     00
D:0X60H:  06      07      08      09     10     00

After execution:
D:0x50H:  06      07      08      09     10     00
D:0X60H:  01      02      03      04     05     00
```

1.3 LARGEST/SMALLEST ELEMENT IN AN ARRAY USING 8051

OBJECTIVE:
TO FIND THE LARGEST/SMALLEST ELEMENT IN AN ARRAY USING 8051

```
PROGRAM TO FIND THE LARGEST NUMBER:
MOV R0,#50H       // Initialize the source memory pointer
MOV R2,#05H       // Initialize  Iteration counter
MOV B, @R0        /* Use B Register to store largest value and initialize
                  it to the first value*/
BACK: MOV A,@R0   /* Get the data from source memory  pointer  and
                  Load into accumulator*/
```

```
CJNE A,B,LOOP     /* Compare the data if not equal, go to relative
                  address(LOOP)*/
LOOP: JC LOOP1    // If carry generates, go to relative address LOOP1
MOV  B,A          // Store larger value into B-register
INC R0            // Increment the source memory pointer
DJNZ R2,BACK      /* Decrement iteration count and if it is not zero,  go
                  to  relative address and repeat the same process until
                  count become zero.*/
SJMP NEXT         // Go to NEXT
LOOP1:INC R0      // Increment the source memory pointer
DJNZ R2,BACK      /* Decrement iteration count and if it is not zero,  go
                  to  relative address and repeat the same process until
                  count become zero.*/
NEXT: MOV 60H,B   /* Store the largest value into memory location 60H.*/
END
```

MEMORY WINDOW:

Before execution:

D:0x50h: 22 AB 3D 44 55 00
D:0x60h: 00 00 00 00 00 00

After execution:

D:0x50h: 22 AB 3D 44 55 00
D:0x60h: AB 00 00 00 00 00

PROGRAM TO FIND THE SMALLEST NUMBER:

```
MOV R0,#50H       // Initialize the source memory pointer
MOV R2,#05H       // Initialize  Iteration counter
MOV B, @R0              /* Use B Register to store smallest value and
                  initialize it to the first value*/
BACK:MOV A,@R0    /* Get the data from source memory  pointer  and Load
                  into accumulator*/
CJNE A,B,LOOP     /* Compare the data if not equal, go to relative
                  address(LOOP)*/
LOOP: JNC LOOP1   // If carry generates, go to relative address LOOP1
MOV  B,A          // Store smaller value into B-register
INC R0            // Increment the source memory pointer
DJNZ R2,BACK      /* Decrement iteration count and if it is not zero,  go
                  to  relative address and repeat the same process until
                  count become zero.*/
SJMP NEXT         // Go to NEXT
LOOP1: INC R0     // Increment the source memory pointer
DJNZ R2,BACK      /* Decrement iteration count and if it is not zero,  go
                  to  relative address and repeat the same process until
                  count become zero.*/
NEXT: MOV 60H,B   /*Store the smallest value into memory location 60H.*/
```

```
END
```

MEMORY WINDOW:

Before execution:

```
      D:0x50H:  22    AB    3D    44    55    00
      D:0X60H:  00    00    00    00    00    00
```

After execution:

```
      D:0x50H:  22    AB    3D    44    55    00
      D:0X60H:  22    00    00    00    00    00
```

1.4 SORTING

OBJECTIVE:
TO ARRANGE N 8-BIT NUMBERS IN ASCENDING ORDER.

```
PROGRAM:
MOV R2, #05H            // Initialize the iteration counter
DEC R2                  // Decrement the iteration count
BACK1: MOV R0, #50H     // Initialize memory pointer1
MOV R1, #51H            // Initialize memory pointer2
MOV A, R2               // Store outer loop count
MOV R3, A               // Store inner loop count
BACK: MOV A,@R0         // Get the data from memory pointer1
MOV B,@R1               // Get the data from memory pointer2
CJNE A, B, LOOP         /* Compare if not equal go to relative address
                        (LOOP)*/
LOOP: JC LOOP1          /* If carry generates, go to relative address
                        (LOOP1)*/
MOV @R0,B               // Exchange the data in memory pointer
MOV @R1, A
LOOP1: INC R0           // Increment the memory pointer1
INC R1                  // Increment the memory pointer2
DJNZ R3, BACK           // Decrement inner loop count if not zero go to
back
DJNZ R2, BACK1          // Decrement outer loop count if not zero go to
back1
END
```

MEMORY WINDOW:

Before execution:

```
      D:0x50H:  06    04    03    07    02    01
```

After execution:

```
      D:0x50H:  01    02    03    04    06    07
```

OBJECTIVE:
TO ARRANGE N 8-BIT NUMBERS IN DESCENDING ORDER.

```
PROGRAM:
MOV R2, #05H            // Initialize the iteration counter
DEC R2                  // Decrement the iteration count
BACK1: MOV R0, #50H     // Initialize memory pointer1
MOV R1, #51H            // Initialize memory pointer2
MOV A, R2               // Store outer loop count
MOV R3, A               // Store inner loop count
BACK: MOV A,@R0         // Get the data from memory pointer1
MOV B,@R1               // Get the data from memory pointer2
CJNE A, B, LOOP // Compare if not equal go to relative address (LOOP)
LOOP: JNC LOOP1 // If carry generates, go to relative address (LOOP1)
MOV @R0,B               // Exchange the data in memory pointer
MOV @R1, A
LOOP1: INC R0           // Increment the memory pointer1
INC R1                  // Increment the memory pointer2
DJNZ R3, BACK     /* Decrement inner loop count, if not zero go to back*/
DJNZ R2, BACK1    /* Decrement outer loop count, if not zero go to back1*/
END
```

MEMORY WINDOW:

Before execution:

D:0x50H: 06 04 03 07 02 01

After execution:

D:0x50H: 07 06 04 03 02 01

WORKSHEET

1. Write an ALP to generate eight Fibonacci numbers using 8051
   - The first term must be zero and second term must be one
   - Add the current term and previous term, store in the next term
   - Repeat the same processes until count become zero.
2. Write an ALP to check the given string of data is palindrome or not
   - The output will be 01 if it is palindrome.
   - The output will be FF if it is not palindrome
3. Explain the difference between the following two instructions:
   a. MOVC A,@R0
   b. MOV A,@R0
4. Circle the invalid instructions.
   a. MOV A,@R1
   b. MOV A,@R2
   c. MOVC A,@R0+DPTR
   d. MOV @R3,A
5. Explain the difference between the following two instructions:
   a. MOV A,40H
   b. MOV A,#40H
6. Explain the difference between the following two instructions:
   a. MOV 40H,A
   b. MOV 40H,#0A
7. Give the RAM address for the following registers.
   a. A = B = R0 = R2 =
   b. PSW = SP = DPL = DPH =

EXPERIMENT NO.2

ARITHMETIC INSTRUCTIONS - ADDITION/SUBTRACTION, MULTIPLICATIO, SQUARE & CUBE (16 BITS ARITHMETIC OPERATIONS)

OBJECTIVE:
TWO UNDERSTAND THE ARITHMETIC OPERATIONS AND PERFORM 8/16 BIT ADDITION/ SUBTRACTION AND MULTIPLICATION

2.1 WRITE AN ALP TO PERFORM 16 BIT ADDITION

```
PROGRAM:
MOV R0,#51H        // Initialize input1 memory pointer
MOV R1,#61H        /* Initialize input2 memory pointer and store output also
                   same */
MOV R2,#02H        // Initialize iteration count
CLR C
BACK: MOV A,@R0    /*Get lower bytes data in first iteration, upper bytes
                   data in second iteration, add them with carry and store
                   in memory pointer2.*/
ADDC A,@R1
MOV @R1,A
DEC R0             // Increment memory pointer1 & 2 to get upper bytes
DEC R1
DJNZ R2,BACK       /* Decrement iteration count and if it is not zero,  go
                   to relative address and repeat the same process until
                   count become zero.*/
JNC FINISH
MOV @R1,#01H
FINISH:SJMP $
END
```

MEMORY WINDOW

Before execution:

D:0x50H:	*FD*	*07*	*00*	*00*	*00*	*00*
D:0X60H:	*FF*	*5F*	*00*	*00*	*00*	*00*

After execution:

D:0x50H:	*FD*	*07*	*00*	*00*	*00*	*00*
D:0X5FH:	*01*	*FC*	*66*	*00*	*00*	*00*

2. 2 WRITE AN ALP TO PERFORM 16 BIT SUBTRACTION

```
PROGRAM:
MOV R0,#51H        //Initialize input1 memory pointer
MOV R1,#61H        /* Initialize input2 memory pointer and store output also
                   same */
MOV R2,#02H        // Initialize iteration count
CLR C
BACK: MOV A,@R0    //Get lower bytes data in first iteration, upper bytes
                   data in second iteration, add them with carry and store
                   in memory pointer2.
SUBB A,@R1
MOV @R1,A
DEC R0             // Increment memory pointer1 & 2 to get upper bytes
DEC R1
DJNZ R2,BACK       /* Decrement iteration count and if it is not zero,  go
                   to relative address and repeat the same process until
                   count become zero.*/
JNC POSITIVE
MOV @R1,#0FFH
JMP FINISH
POSITIVE: MOV @R1,#00H
FINISH: SJMP $
END
Eg. FAF4 - 02F5 = F7FF (ANSWER IS POSITIVE)
MEMORY WINDOW
Before execution:
D:0x50H:  FA     F4       00      00      00      00
D:0X60H:  02     F5       00      00      00      00
After execution:
D:0x50H:  FA     F4       00      00      00      00
D:0X60H:  F7     FF       00      00      00      00
Eg. 0025 – 0AF6 = FFF52F (ANSWER IS NEGATIVE)
Before execution:
D:0x50H:  00     25       00      00      00      00
D:0X60H:  0A     F6       00      00      00      00
After execution:
D:0x50H:  00     25       00      00      00      00
D:0x5FH:  FF     F5       2F      00      00      00
```

2. 3 WRITE AN ALP TO PERFORM MULTIPLICATION (16-BIT BY 16-BIT)

First number will be in R6 and R7 while second number will be in R4 and R5. The result will be in R0, R1, R2 and R3.

Eg:

```
R6 R7 = 15 FD
R4 R5 = A2 4B
R0 R1 R2 R3 = 0D F0 8B 1F
PROGRAM:
MOV R6,#0FFH      //FFFF X FFFF = FFFE 0001
MOV R7,#0FFH      // input the multiplicand
MOV R4,#0FFH      // input the multiplier
MOV R5,#0FFH
//Multiply R5 by R7
MOV A,R5          // Move the R5 into the Accumulator
MOV B,R7          // Move R7 into B
MUL AB            // Multiply the two values
MOV R2,B          // Move B (the high-byte) into R2
MOV R3,A          // Move A (the low-byte) into R3
//Multiply R5 by R6
MOV A,R5          // Move R5 back into the Accumulator
MOV B,R6          // Move R6 into B
MUL AB            // Multiply the two values
ADD A,R2          // Add the low-byte into the value already in R2
MOV R2,A          // Move the resulting value back into R2
MOV A,B           // Move the high-byte into the accumulator
ADDC A,#00h       // Add zero (plus the carry, if any)
MOV R1,A          // Move the resulting answer into R1
MOV A,#00h        // Load the accumulator with  zero
ADDC A,#00h       // Add zero (plus the carry, if any)
MOV R0,A          // Move the resulting answer to R0.
//Multiply R4 by R7
MOV A,R4          // Move R4 into the Accumulator
MOV B,R7          // Move R7 into B
MUL AB            // Multiply the two values
ADD A,R2          // Add the low-byte into the value already in R2
MOV R2,A          // Move the resulting value back into R2
```

```
MOV A,B           // Move the high-byte into the accumulator
ADDC A,R1         // Add the current value of R1 (plus any carry)
MOV R1,A          // Move the resulting answer into R1.
MOV A,#00h        // Load the accumulator with zero
ADDC A,R0         // Add the current value of R0 (plus any carry)
MOV R0,A          // Move the resulting answer to R1.
//Multiply R4 by R6
MOV A,R4          // Move R4 back into the Accumulator
MOV B,R6          // Move R6 into B
MUL AB            // Multiply the two values
ADD A,R1          // Add the low-byte into the value already in R1
MOV R1,A          // Move the resulting value back into R1
MOV A,B           // Move the high-byte into the accumulator
ADDC A,R0         // Add it to the value already in R0 (plus any carry)
MOV R0,A          // Move the resulting answer back to R0
// answer is now in R0, R1, R2, and R3
SJMP $
END

RESULT
REGISTER VALUES:
R6 R7 = FF FF
R4 R5 = FF FF
R0 R1 R2 R3 = FF FE 00 01
```

2. 3 WRITE AN ALP TO PERFORM DIVISION (16-BIT BY 16-BIT)

First number will be in R1 and R0 while second number will be in R3 and R2. The result will be in R2 and R3. .

Eg:

```
R1 R0 = D7 4E
R3 R2 = 00 D9
R3 R2 = 00 FE
PROGRAM:
MOV R1,0D7H
MOV R0,4EH
MOV R3,00H
MOV R2,0D9H
div16_16:
```

```
CLR C       // Clear carry initially
MOV R4,#00h // Clear R4 working variable initially
MOV R5,#00h // CLear R5 working variable initially
MOV B,#00h  /* Clear B since B will count the number of left-shifted bits*/
div1:
INC B       // Increment counter for each left shift
MOV A,R2    // Move the current divisor low byte into the accumulator
RLC A /* Shift low-byte left, rotate through carry to apply highest bit to
high-byte*/
MOV R2,A    // Save the updated divisor low-byte
MOV A,R3    /* Move the current divisor high byte into the accumulator*/
RLC A       // Shift high-byte left high, rotating in carry from low-byte
MOV R3,A    // Save the updated divisor high-byte
JNC div1    // Repeat until carry flag is set from high-byte
div2:             // Shift right the divisor
MOV A,R3    // Move high-byte of divisor into accumulator
RRC A       // Rotate high-byte of divisor right and into carry
MOV R3,A    // Save updated value of high-byte of divisor
MOV A,R2    // Move low-byte of divisor into accumulator
RRC A       // Rotate low-byte of divisor right, with carry from high-byte
MOV R2,A    // Save updated value of low-byte of divisor
CLR C       // Clear carry, we don't need it anymore
MOV 07h,R1  // Make a safe copy of the dividend high-byte
MOV 06h,R0  // Make a safe copy of the dividend low-byte
MOV A,R0    // Move low-byte of dividend into accumulator
SUBB A,R2   /* Dividend - shifted divisor = result bit (no factor, only 0
or 1)*/
MOV R0,A    // Save updated dividend
MOV A,R1    // Move high-byte of dividend into accumulator
SUBB A,R3   /* Subtract high-byte of divisor (all together 16-bit
subtraction)*/
MOV R1,A    // Save updated high-byte back in high-byte of divisor
JNC div3    // If carry flag is NOT set, result is 1
MOV R1,07h  /* Otherwise result is 0, save copy of divisor to undo
subtraction*/
MOV R0,06h
div3:
CPL C       // Invert carry, so it can be directly copied into result
MOV A,R4
RLC A       // Shift carry flag into temporary result
```

```
MOV R4,A
MOV A,R5
RLC A
MOV R5,A
DJNZ B,div2 // Now count backwards and repeat until "B" is zero
MOV R3,05h  // Move result to R3/R2
MOV R2,04h  // Move result to R3/R2
SJMP $
END
RESULT
REGISTER VALUES:
R1 R0 = D7 FE     // D7FE/D9 = FE
R3 R2 = 00 D9
R3 R2 = 00 FE
```

WORKSHEET

1. Explain the difference between the ADD and ADDC instructions.
2. Show how to perform the subtraction: 29H - 21H.
4. True or False. "DA A" must be used for adding BCD data only.
5. Can we use the "DA A" instruction to convert data such as 9CH into BCD without first performing an ADD instruction? Explain your answer.
6. Show a simple program to add 2345H and 56F8H.
7. Show a simple program to subtract 2345H from 56F8H.

EXPERIMENT NO.3

COUNTERS

OBJECTIVES:
TWO UNDERSTAND THE SIMULATION OF BINARY/BCD UP/DOWN COUNTERS AND TO KNOW THE CONCEPTS OF SUBROUTINES

3. 1 WRITE A PROGRAM TO REALIZE A BINARY UP COUNTER

```
ORG 0000H          // Organization of code memory from 0000h
      CLR 50H      // Clear upper byte counter
      CLR 51H      // Clear lower byte counter
UP:   ACALL DELAY /* Call the subroutine to provide delay between two
counter value*/
// LOWER BYTE COUNTER
      MOV A, 51H  // Get the current lower counter
      ADD A, #01H // Add 01h with previous value to get next counter
      MOV 51H,A   //  Store the counter  in lower byte
JNZ UP      /* If lower count value not zero, go to relative address (UP)*/
// UPPER BYTE COUNTER
MOV A,50H          /* If lower byte reaches zero, get the current upper
                   counter*/
      ADD A,#01H  // Add 99h to previous value to get next counter
      MOV 50H,A   // Store the counter in upper byte
      JNZ UP      /* If upper count value not zero, go to relative address
                  (UP)*/
      SJMP UP     // Repeat this counter until stop running

// Provide delay between two counter value
 DELAY: MOV DPTR, #04FFH      // Initialize the memory pointer
    L2: INC DPTR              // Increment the memory pointer
        MOV A, DPL      // Add higher byte and lower byte address
        ORL A, DPH
JNZ L2                  // If it is not zero,go to relative address  (L2)
        RET             // Return to main program
        END
RESULT
During execution:D:0x50H: (00 to FFH) (00 to FFH) 00 00 00 00
```

3. 1 WRITE A PROGRAM TO REALIZE BINARY DOWN COUNTER

```
      MOV 50H,#0FFH     // Initialize upper byte of  HEX counter with FFH
      MOV 51H,#0FFH     // Initialize lower byte of  HEX counter with FFH
UP:   ACALL DELAY       // Call the subroutine to provide delay
                        //between two Counter value
// LOWER BYTE COUNTER
      DEC 51H     // Decrement lower byte counter
      MOV A, 51H  // Store the lower byte counter into
                  //accumulator
      JNZ UP      // If lower count value not zero, go to
                  //relative address (UP)
// UPPER BYTE COUNTER
      DEC 50H     //Decrement upper byte counter
      SJMP UP     // Repeat this counter until stop running
// Provide delay between two counter value
DELAY:MOV DPTR, #04FFH  // Initialize the memory pointer
L2:   INC DPTR    // Increment the memory pointer
      MOV A, DPL  // Add higher byte and lower byte address
      ORL A, DPH
      JNZ L2      // If it is not zero, go to relative
                  // address(L2)
      RET         // Return to main program
      END
```

RESULT

During execution:

D:0x50H : (FFH to 00H) (FFH to 00H) 00 00 00 00

3. 2 WRITE A PROGRAM TO REALIZE A BCD COUNTER

WRITE AN ALP TO PERFORM BCD UP COUNTER

```
      ORG 0000H         // Organization of code memory from 0000h
      CLR 50H           // Clear upper byte counter
      CLR 51H           // Clear lower byte counter
UP:   ACALL DELAY       // Call the subroutine to provide delay between
                        // two counter value
```

```
// LOWER BYTE COUNTER
      MOV A, 51H        // Get the current lower counter
      ADD A, #01H       // Add 01h to previous value to get next counter
      DA A              // Convert hex value to decimal
      MOV 51H, A        // Store the counter in lower byte
      JNZ UP            // If lower count value not zero, go to
                        // relative address (UP)
// UPPER BYTE COUNTER
      MOV A, 50H        // If lower byte reaches zero, get the current
                        // upper counter
      ADD A, #01H       // Add 99h to previous value to get next counter
      DA A              // Convert hex value to decimal
      MOV 50H, A        // Store the counter in upper byte
      JNZ UP            // If upper count value not zero, go to
                        // relative address (UP)
      SJMP UP           // Repeat this counter until stop running
// Provide delay between two counter value
      DELAY: MOV DPTR, #04FFH // Initialize the memory pointer
      L2: INC DPTR      // Increment the memory pointer
      MOV A, DPL        // Add higher byte and lower byte address
      ORL A, DPH
      JNZ L2            // If it is not zero, go to relative address (L2)
      RET               // Return to main program
      END
```

RESULT

During execution: D:0x50H :(00 to 99H) (00 to 99H) 00 00 00 00

WRITE AN ALP TO PERFORM BCD DOWN COUNTER

```
ORG 0000H         // Organization of code memory from 0000h
      MOV 50H, #99H     //Initialize upper byte of counter with 99h
      MOV 51H, #99H     //Initialize lower byte of counter with 99h
UP:   ACALL DELAY       /* Call the subroutine to provide delay between two
      counter value*/
// LOWER BYTE COUNTER
      MOV A, 51H        // Get the current lower counter
      ADD A, #99H       /* Add 99h to previous value to get next counter*/
      DA A              // Convert hex value to decimal
```

```
      MOV 51H,A          //  Store the counter  in lower byte
      JNZ UP             /* If lower count value not zero, go to relative
      address (UP)*/
// UPPER BYTE COUNTER
      MOV A, 50H        // If lower byte reaches zero, get the current
                        //upper counter
      ADD A, #99H       /* Add 99h to previous value to get next counter*/
      DA A              // Convert hex value to decimal
      MOV 50H,A         // Store the counter in upper byte
      JNZ UP            /* If upper count value not zero, go to relative
      address (UP)*/
      SJMP UP          // Repeat this counter until stop running
// provide delay between two counter value
DELAY:MOV DPTR, #04FFH  // Initialize the memory pointer
L2:   INC DPTR          // Increment the memory pointer
      MOV A, DPL        // Add higher byte and lower byte address
      ORL A, DPH
      JNZ L2            // If it is not zero, go to relative
                        // address(L2)
      RET               // Return to main program
      END
```

RESULT

During Execution: D:0x50H :(99H to 00H) (99H to 00H) 00 00 00 00

Assignment:

Following is a delay subroutine. Find how much delay this subroutine provides. Insert the delay appropriately inside the BCD COUNTER program so that each count should happen after twice the delay

```
DELAY SUBROUTINE:
DELAY:      NOP
            MOV R2,#25
H0:         MOV R3,#255
HI:         MOV R4,#255
            DJNZ R4,$
            DJNZ R3,HI
            DJNZ R2,H0
            RET
```

EXPERIMENT NO.4

BOOLEAN & LOGICAL INSTRUCTIONS (BIT MANIPULATIONS)

OBJECTIVES:
TWO UNDERSTAND THE BASIC LOGICAL BIT AND BYTE OPERATIONS

4. 1. EXAMPLES FOR LOGICAL BYTE OPERATIONS

```
ORG 00H
MOV R0, #34H

MOV A, R0
ANL A, #0FH       //and logical operation
MOV P1, A

MOV A, R0
ORL A, #0FH       //or logical operations
MOV P1, A

MOV A, R0
XRL A, #0FH       //exclusive or logical operations
MOV P1, A

MOV A, R0
CPL A             //complement logical operations
MOV P1, A

MOV A, R0
CLR A             //clear logical operations
MOV P1, A

MOV A, R0
RR A              //rotate right logical operations
RR A
RR A
RR A
MOV P1, A
```

```
MOV A, R0
RL A              //rotate left logical operations
RL A
RL A
RL A
MOV P1, A

MOV A, R0
SETB C
RRC A             //rotate right with carry logical operations
RRC A
RRC A
RRC A
MOV P1, A

MOV A, R0
RLC A             //rotate left with carry logical operations
RLC A
RLC A
RLC A
MOV P1, A
SJMP $
END
```

4. 2 EXAMPLES OF LOGICAL BIT OPERATIONS

```
SETB 01H    //set bit addressable memory 01h
CLR 01H     //clear bit addressable memory 01h
CPL 01H     //compliment bit addressable memory 01h
MOV C, 01H  /*move bit addressable memory location 01h's content to
            carry*/
MOV 07FH,C  //move carry to bit addressable memory location 01h
SETB 01H

ANL C, 01H  /*logically and bit addressable memory 01h content with
carry*/
ORL C, 01H  /*logically or bit addressable memory 01h content with
            carry*/
SJMP $
```

```
      END
```

4.3. TWO OUT OF FIVE CODE

This is a program will check the data available at port 1 is a "2_out_of_5_code" or not. If yes it will send f0 to the port0 otherwise it will send 0f to port 0.
This program demonstrates the use of some LOGICAL instruction and CONDITIONAL/UNCONDITIONAL JUMPS

```
Initializing ports and counter.
      MOV P1, #0FFH
      MOV P0, #00H
      MOV R2, #05H
/*Initially Checking whether any of the last three Most Significant Bits
are high or not. If it is high it is not a "2_OUT_OF_5_CODE"*/
MAIN: MOV A, P1
      MOV R0, A
      ORL A, #01FH
      CJNE A, #01FH, NOTOF
/*If all the last three MSB's are zero then checking for how many 1's are
present in the remaining five bits.*/
CHECK:MOV A, R0
      AGAIN:RRC A
      JNC LOOP
      INC R1
LOOP: DEC R2
      CJNE R2, #00, AGAIN
/*After completing five times rotation,  R1 contains the number of 1's in
last five bits. If the number is two it isa"2_OUT_OF_5_CODE".*/
      CJNE R1, #02, NOTOF
      MOV P0, #0F0H     //Set upper Nibble of P0 high if it is a
                        //"2_OUT_OF_5_CODE"
      SJMP HERE
NOTOF:MOV P0, #00FH     //Set lower Nibble of P0 high if it is not a
HERE: SJMP HERE         //"2_OUT_OF_5_CODE"
      END
```

4. 4 BOOLEAN EXPRESSIONS

1. WRITE PROGRAMS TO REALIZE SOME BOOLEAN EXPRESSIONS

TABLE : 1
BIT ADDRESSIBLE MEMORY LOCATIONS AND CORRESPONGING ADDRESSES

BIT	BYTE	BIT	BYTE	BIT	BYTE	BIT	BYTE
00	20.0	10	21.0	20	24.0	30	26.0
01	20.1	11	22.1	21	24.1	31	26.1
02	20.2	12	22.2	22	24.2	32	26.2
03	20.3	13	22.3	23	24.3	33	26.3
04	20.4	14	22.4	24	24.4	34	26.4
05	20.5	15	22.5	25	24.5	35	26.5
06	20.6	16	22.6	26	24.6	36	26.6
07	20.7	17	22.7	27	24.7	37	26.7
08	20.0	18	22.0	28	25.0	38	27.0
09	21.1	19	23.1	29	25.1	39	27.1
0A	21.2	1A	23.2	2A	25.2	3A	27.2
0B	21.3	1B	23.3	2B	25.3	3B	27.3
0C	21.4	1C	23.4	2C	25.4	3C	27.4
0D	21.5	1D	23.5	2D	25.5	3D	27.5
0E	21.6	1E	23.6	2E	25.6	3E	27.6
0F	21.7	1F	23.7	2F	25.7	3F	27.7

```
// VARIABLE DECLARATION
      X EQU 10H
      Y EQU 11H
      Z EQU 12H

      NOT_X EQU 20H
      NOT_Y EQU 21H
      NOT_Z EQU 22H

      OUTPUT EQU P0.0

//INITIALIZING PORTS P1 AS INPUT AND P2 AS OUTPUT PORT

      MOV P1, #0FFH
      MOV P0, #00H

//INTIALIZING X,Y,Z AND ITS COMPLEMETS AFTER READING THE VALUES FROM PORT 1

      MOV C, P1.0
```

```
      MOV X, C
      CPL C
      MOV NOT_X, C

      MOV C, P1.1
      MOV Y, C
      CPL C
      MOV NOT_Y, C

      MOV C, P1.2
      MOV Z, C
      CPL C
      MOV NOT_Z, C

//EXPRESSION OUTPUT = X+Y+Z
      CLR C
      MOV C, X
      ORL C, Y
      ORL C, Z
      MOV OUTPUT, C

//EXPRESSION OUTPUT = XYZ
      CLR C
      MOV C, X
      ANL C, Y
      ANL C, Z
      MOV OUTPUT, C

//EXPRESSION OUTPUT = XY+ Z
      CLR C
      MOV C, X
      ANL C, Y
      ORL C, Z
      MOV OUTPUT, C

//EXPRESSION OUTPUT = X+ YZ
      CLR C
      MOV C, Y
      ANL C, Z
```

```
      ORL C, X
      MOV OUTPUT, C

//EXPRESSION OUTPUT =!X+ !Y+ !Z
      CLR C
      MOV C, NOT_X
      ORL C, NOT_Y
      ORL C, NOT_Z
      MOV OUTPUT, C


//EXPRESSION OUTPUT =!(X+Y)+ Z
      CLR C
      MOV C, X
      ORL C, Y
      CPL C
      ORL C, Z
      MOV OUTPUT, C

//EXPRESSION OUTPUT =!(X+Y+Z)
      MOV C, X
      ORL C, Y
      ORL C, Z
      CPL C
      MOV OUTPUT, C

//EXPRESSION OUTPUT = XYZ + !(XY)Z + !Y!Z
      CLR C
      MOV C, X
      ANL C, Y
      ANL C, Z
      MOV 00H, C
      MOV C, X
      ANL C, Y
      CPL C
      ANL C, Z
      MOV 01H, C
      MOV C, NOT_Y
      ANL C, NOT_Z
```

```
      ORL C, 01H
      ORL C, 00H
      MOV OUTPUT, C

HERE: SJMP HERE
      END
```

NOTE:
Give minimum 4 input combinations and verify the results for each expression

***Assignment**: Realize the function WX+WYX + YZ*

EXPERIMENT NO.5

CONDITIONAL CALL AND RETURN

OBJECTIVES:
TO DEMONSTRATE CONDITIONAL BIT JUMP, CONDITIONAL BYTE JUMP, UNCONDITIONAL JUMP, CALL and RETURN instructions

This program keeps checking P1.0 for low signal. Once a low signal arrives, program checks for AAh at PORT 0. If the data is AAh a counter will start.

```
      ORG  00H
/*INITIALIZING THE COUNTER AND PORTS*/
MAIN: MOV P0, #0FFH
      MOV P1, #0FFH
      MOV P2, #00H
      MOV R0, #00H
/*WAITING FOR THE ARRIVAL OF LOW SIGNAL AT PORT 1.0*/
      WAIT: JB P1.0, WAIT     // CONDITIONAL BIT JUMP
      ACALL VERIFY_DATA       // CONDITIONAL CALL
      SJMP WAIT               // UNCONDITIONAL JUMP
                              //CHECKING THE DATA AT P0 FOR 0AAh
VERIFY_DATA:MOV A, P0
      CJNE A, #0AAH, RETURN   // CONDITIONAL BYTE JUMP
      INC R0
      MOV P2, R0
      ACALL DELAY             // UNCONDITIONAL CALL
RETURN:RET                          // RETURN

/*JUST A DELAY SUBROUTINE ☺*/
DELAY:NOP
      MOV R2,#25
H0:   MOV R3,#255
HI:   MOV R4,#255
      DJNZ R4,$               // CONITIONAL BYTE JUMPS
      DJNZ R3,HI
      DJNZ R2,H0
      RET                     // RETURN
END
```

Worksheet

1. Upon reset, all the ports of the 8051 are configured as _____ (input, output).
2. To make all the bits of a port an input port we must write ____ hex to it.
3. Which ports of the 8051 are bits addressable?
4. What does it mean for port to be "read-modify-write"?
5. Write a program to monitor P2.4 continuously. When it becomes low, it sends 55H to P1.

EXPERIMENT NO.6

CODE CONVERSION

OBJECTIVES:
TWO PERFORM CONVERSIONS OF DIFFERENT NUMBER SYSTEM REPRESENTATIONS BY MAKING USE OF LOGICAL INSTRUCTIONS.

6. 1 BCD – ASCII

```
; register a has packed BCD
; Program to convert BCD to '2' ASCII numbers
      ORG 0H
      MOV A, #29H          //mov 29 to A( packed BCD)
      MOV R2, A            //copy of A (BCD DATA)in R2
      ANL A, #0FH          //mask the upper nibble A=09h
      ORL A, #30H          //maske it as ASCII to A=39h(ASCII char 09h
      MOV R6,A             //save it in R6
      MOV A, R2            //A=29, get original data
      ANL A, #0F0H         //mask the lower nibble A=20h
      RR A                 //rotate right
      RR A                 //rotate right
      RR A                 //rotate right
      RR A                 //rotate right A=02h
      ORL A, #30H          //A=32h, ASCII char 2
      MOV R2,A             //save ASCII to char in R2
      MOV P1,A
      MOV P2,R6
      SJMP $
      END
```

6. 2. ASCII – DECIMAL

```
      ORG 0
      MOV R5, #"4"      // LOADING R5 WITH ASCII EQUIVALENT OF 4 (IE, 34)
      MOV R6, #"7"      // LOADING R6 WITH ASCII EQUIVALENT OF 7 (IE, 37)
      MOV A, R5
      ANL A, #0FH       // MASK LOWER NIBBLE
      MOV R5, A
      MOV A, R6
      ANL A, #0FH       // MASK UPPER NIBBLE
      MOV R6, A
      MOV A, R5
      RL A
      RL A
      RL A
      RL A
      ORL A, R6         // PACKING
      MOV P1,A
      SJMP $
      END
```

6. 3. DECIMAL – ASCII

(Conversion of Decimal number 0-9 to corresponding ASCII equivalent)

```
      RAM_ADDR EQU 40H        //initializing addresses
      ASC_RSULT EQU 50H       //for storing data and result

      ORG 00H
MAIN: MOV R0,#RAM_ADDR
      MOV R1,#ASC_RSULT
      MOV R2,#0AH
BACK:   MOV A,@R0             //stored decimal values
      ORL A,#30H              //are fetched ,they are 'OR'ed with
      MOV @R1,A               //30 and results are placed at memory
      INC R0                  //pointed by R1 pointer
      INC R1
      DJNZ R2,BACK
      MOV R1,#ASC_RSULT

      MOV R2,#0AH             //results are verified on
NEXT: MOV A,@R1               //PORT 1
      MOV P1,A
      INC R1
      DJNZ R2,NEXT
      SJMP $
      END
```

6. 4. HEX - DECIMAL

(Verify the Results of Minimum Five Input Combinations)

```
      ORG 0000          //start
MAIN: MOV A,#0FFH       //load FF to ACC
      MOV P1,A          // and make P1 an input PORT
      MOV A,P1          //take hex data from input port P1
      MOV B,#10         //divide by 10 remainder is stored in B
      DIV AB
      MOV R7,B          //unit place of decimal is put to R7/
      MOV B,#10
      DIV AB                  //divide again
      MOV R6,B          //tens place of decmial is put to R6
      MOV R5,A          //hundred's pplace  is put to R5
      LJMP MAIN
      END               // end
```

6. 5. DECIMAL – HEX

```
      ORG 00            //  start
      MOV R3,#002
      MOV R4,#00
      MOV A,#251       //decmial i/p is given
      MOV B,#02        //divide the no by 2
      CLR C
      DIV AB
      MOV R5,A         // ACC value is lost after div ,so move it to R5
      MOV A,B           //MOV value of B to ACC
      ORL A,R4          // ORing acc value with R4
      MOV R4,A          // store the ORed value in R4

      MOV B,#02
      MOV A,R5
      DIV AB
      MOV R5,A
      MOV A,B
      RL A              //rotate the ACC left once ,to get first bit
      ORL A,R4
      MOV R4,A

      MOV B,#02
      MOV A,R5
      DIV AB
      MOV R5,A
      MOV A,B
      RL A              // rotate the ACC twice, to get the third bit
      RL A
      ORL A,R4
      MOV R4,A

      MOV B,#02
      MOV A,R5
      DIV AB
      MOV R5,A
      MOV A,B
      RL A              //rotate the ACC thrice to get the fourth bit
      RL A
      RL A
      ORL A,R4
      MOV R4,A

      MOV A,R5
      SWAP A
      ORL A,R4
      MOV P1,A          //display the final result on port1
HERE: SJMP  HERE        //end of the program
      END
```

WORKSHEET

1. Find the value in A, the accumulator, after the following code.

```
MOV   A, #45H
RR    A
RR    A
RR    A
A =           in hex
```

2. Find the value in A, the accumulator, after the following code.

```
MOV   A, #45H
RL    A
RL    A
RL    A
A =           in hex
```

3. In the absence of the "SWAP A" instruction, how would you perform the operation?
4. Can the SWAP instruction work on any register?
5. Find the value in A after the following code.

```
CLR   A
XRL   A,#0FFH
A =           in hex
```

6. Find the value in A after the following code.

```
CLR   A
CPL   A
XRL   A,#0FFH
A =           in hex
```

EXPERIMENT NO.7

PROGRAMS TO GENERATE DELAY USING SERIAL PORT AND ON-CHIP TIMER / COUNTER

OBJECTIVES:
TWO UNDERSTAND THE BASICS OF SERIAL PORT COMMUNICATION

```
CR EQU 0DH        //CLEAR LINE
LF EQU 0AH        // LINE FEED
ORG 0             //Initialize the Special Function Registers Required
                  //for Serial Data Transmission
/*Clear the Serial Window(Screen) and return the curser to beginning of the
    line*/

      MOV A,#0CH  // clear screen
      ACALL SENDsr
      MOV A,#LF
      ACALL SENDsr

/*Retrieve The Data to be transmitted from the code memory*/

      MOV  DPTR,#MYDATA0      // load pointer for message
H_1:  CLR  A
      MOVC  A,@A+DPTR         // get the character
      JZ  HERE                // if last character get out
      ACALL SENDsr            // otherwise call transfer
      ACALL DELAY
      INC  DPTR               // next one
      SJMP H_1                //stay in loop

                              // Serial data transfer. ACC has the data

SENDsr: MOV TMOD, #20H        // timer 1, mode 2(auto-reload)
      MOV TH1, #-3            // 9600 baud rate
      MOV SCON,#50H           // 8-bit,1 stop, REN enabled
      SETB TR1                // start timer 1
      MOV SBUF,A              // load the data
      JNB TI,$                // stay here until last bit gone
      CLR TI                  // get ready for next char
      RET                     // return to caller
                              // Receive data serially in Acc
/*Delay Subroutine*/

DELAY:      MOV R0,#05H
DELAY1:     MOV TMOD,#10H
            MOV TL1,#00H
            MOV TH1,#00H
            SETB TR1
```

```
            JNB TF1,$
            CLR TR1
            CLR TF1
            DJNZ R0,DELAY1
            RET


// The message

MYDATA0:    DB   CR,LF,"Hello World" ,0
HERE:       SJMP HERE
      END
```

NOTE: MODIFY THIS PROGRAM TO SEND THE STRING IN A LOOP

GENERAL WORKSHEET

1. Give the highest single digit for each of the number systems: decimal, binary, and hex.
2. Which of the following cannot be a number in base-2? Give the reason.
   (a) 11001 (b) 113 (c) 10001
3. What is the highest 8-bit number?
   (a) In binary: (b) In hex:
4. What is the highest 16-bit number?
   (a) In binary: (b) In hex:
5. Convert binary 100000 to decimal and hex.
   (a) Decimal: (b) Hex:
6. Convert hex number BAAD to binary and decimal.
   (a) Binary: (b) Decimal:
7. Find the value of the CY flag after the execution of the following code.

```
(a)   MOV  A,#85H
      ADD  A,#92H
(b)   MOV  A,#15H
      ADD  A,#72H
(c)   MOV  A,#0F5H
      ADD  A,#52H
(d)   MOV A,#0FF
      INC A
```

8. Upon reset, what is the value in the SP register?
9. Upon pushing data onto the stack, the SP register is _____ (decremented, incremented).
10. Upon popping data from the stack, the SP register is _____ (decremented, incremented).
11. Can you change the value of the SP register? If yes, explain why you would want to do that.
12. The stack uses the same area of RAM as bank _____.
13. Indicate the size (8- or 16-bit) of each of the following registers.

    PC = A= B=
    R0= R1= R2= R7=

14. For Question 1, indicate the largest value (in decimal) that each register can contain.

    PC = A= B=
    R0= R1= R2= R7=

15. For Question 1, indicate the largest value (in hex) that each register can contain.

    PC = A= B=
    R0= R1= R2= R7=

16. Who generates each of the following files and what is the use of each.

    .asm .lst .obj .abs .hex

PART II
INTERFACING

PORT DETAILS

DC/STEPPER MOTOR PORT		DIP SWITCH/ EEPROM/RTC PORT		
1	P0.0	1	S1	P2.0
2	P0.1	2	S2	P2.1
3	P0.2	3	S3	P2.2
4	P0.3	4	S4	P2.3
5	P0.4	5	S5	P2.4
6	P0.5	6	S6	P2.5
7	P0.6	7	S7	P2.6
8	P0.7	8	S8	P2.7

DAC PORT		KEY BOARD PORT		
1	P0.0	1	R0	P2.0
2	P0.1	2	R1	P2.1
3	P0.2	3	R2	P2.2
4	P0.3	4	R3	P2.3
5	P0.4	5	C0	P1.0
6	P0.5	6	C1	P1.1
7	P0.6	7	C2	P1.2
8	P0.7	8	C3	P1.3

LCD PORT			7 – SEGMENT PORT			ADC PORT		
1	P3.2	RS	1	P0.0	A	1	P1.0	WR
2	P3.3	R/W	2	P0.1	B	2	P1.1	RD
3	P3.4	EN	3	P0.2	C	3	P1.2	CS
4	P0.0	DAT 0	4	P0.3	D	4	P1.3	INTR
5	P0.1	DAT 1	5	P0.4	E	5	P2.0	DATA7
6	P0.2	DAT 2	6	P0.5	F	6	P2.1	DATA6
7	P0.3	DAT 3	7	P0.6	G	7	P2.2	DATA5
8	P0.4	DAT 4	8		DP	8	P2.3	DATA4
9	P0.5	DAT 5	9	P3.2	DIS-1	9	P2.4	DATA3
10	P0.6	DAT 6	10	P3.3	DIS-2	10	P2.5	DATA2
11	P0.7	DAT 7	11	P3.4	DIS-3	11	P2.6	DATA1
12			12	P3.5	DIS-4	12	P2.7	DATA0

SERIAL PORT	
RXD	P3.0
TXD	P3.1
INT0	P3.2
INT1	P3.3

PORT DETAILS FOR INTERFACING THE MICROCONTROLLER CARD WITH OTHER PERIPHERALS

EXPERIMENT NO.8

SIMPLE CALCULATOR USING SIX DIGIT SEVEN SEGMENT DISPLAY AND HEX KEYBOARD INTERFACE TO 8051

OBJECTIVE

REALIZING A SIMPLE CALCULATOR USING MICROCONTROLLER. ADDITION, SUBTRACTION, MULTIPLICATION AND DIVISION IS PERFORMED ON DECIMAL NUMBERS

Hardware:

Microcontroller	89s8252	
Crystal Freq	11.0592	

I/O Port configuration

Port 0	output;	7 Segment Display
Port 2	input:	Keypad

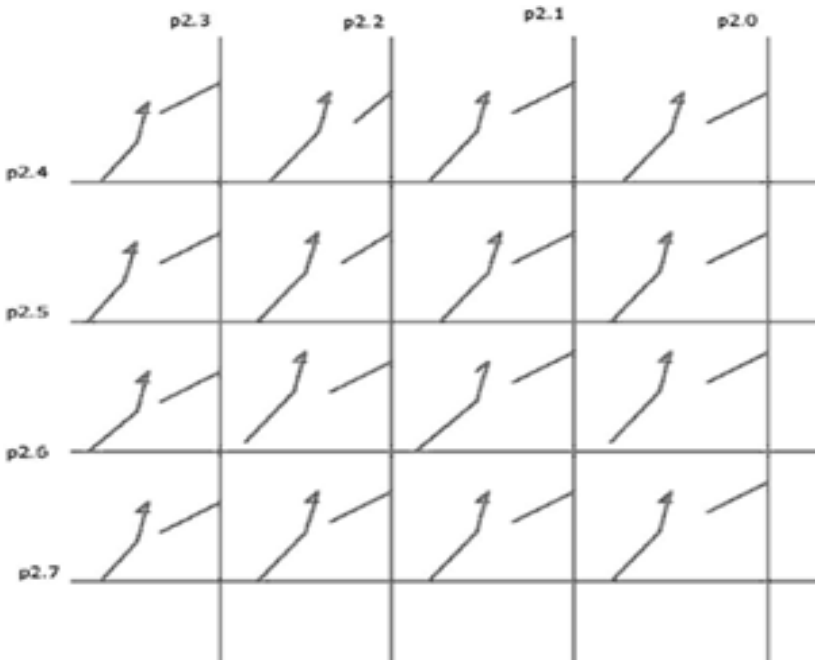


Figure 1. Hex Keypad Port Details

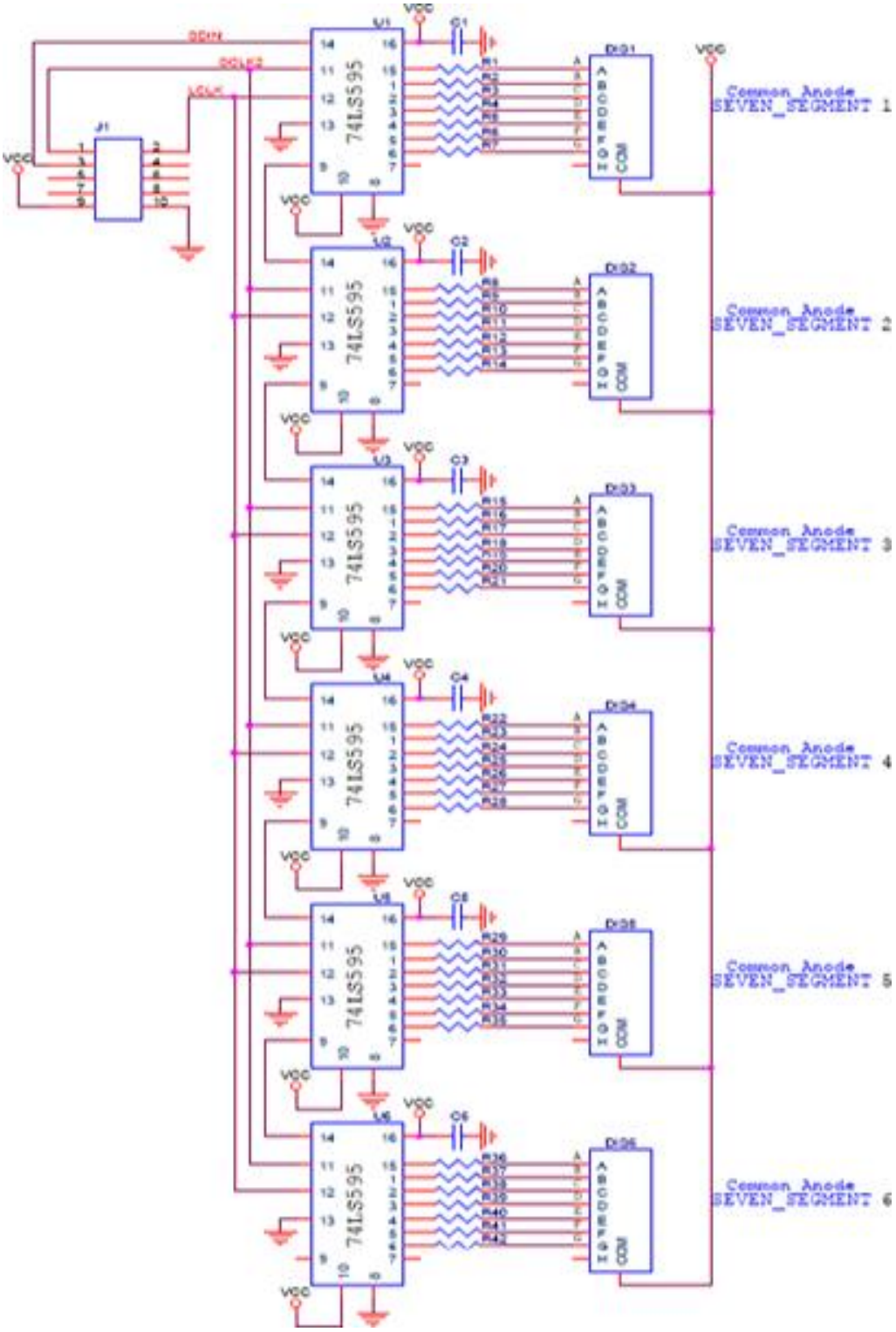


Figure 2. Seven Segment Display

PROGRAM

```
#include <REG51.H>
sbit srck = P1^1 ;
sbit ser  = P1^2 ;
sbit rck  = P1^0 ;

//--------------Funtion declaration

void display(unsigned char);    //LED Display Routine
void convert_display(unsigned char);
void MSDelay (unsigned int);    //Delay Routine
void Clear (void);
unsigned char getkey ();        //KeyBoard Routine

unsigned char key=0xFF;         //Recieved KeyStroke

unsigned char idata keypad[4][4]= { '1','2','3','/',
                                    '4','5','6','*',
                                    '7','8','9','-',
                                    'C','0','C','+',};
unsigned char idata disp[10] =
                {0x40,0xcf,0xa4,0x30,0x19,0x12,0x02,0xf8,0x00,0x10 };
void main()
      {
        unsigned char Data1,Data2,Ans,Funct;
        Clear();
        while(1)
            {
            do
                  {
                        getkey ();    //get data from matrix key pad
                        MSDelay(10);
                        Data1 = key;
                        key = key & 0xF0;
                        }while (key!=0x30);
            display(disp[key&0x0F]);
            do
                  {
                        getkey ();    //get data from matrix key pad
                        MSDelay(10);
                        Funct = key;
                        key = key & 0xF0;
                  }while (key!=0x20);
            Clear();
            do
                  {
                        getkey ();  //get data from matrix key pad
                        MSDelay(10);
                        Data2 = key;
                        key = key & 0xF0;
                  }while (key!=0x30);
```

```
            display(disp[key&0x0F]);

            Data1= Data1 & 0x0F;
            Data2= Data2 & 0x0F;
            switch (Funct)          //loop to respective subroutine
                              {
                                    case ('+'):
                                    {
                                    Ans = Data1 + Data2;
                                    convert_display(Ans);
                                    break;
                                    }
                                    case ('-'):
                                    {
                                    Ans = Data1 - Data2;
                                    convert_display(Ans);

                                    break;
                                    }
                                    case ('*'):
                                    {
                                    Ans = Data1 * Data2;
                                    convert_display(Ans);

                                    break;
                                    }
                                    case ('/'):
                                    {
                                    Ans = Data1 / Data2;
                                    convert_display(Ans);

                                    break;
                                    }
                              }
            }
      }

void Clear (void)
      {
       unsigned int x;
       for (x=0;x<6;x++)
            {
             display(0xFF);
            }
      }
void MSDelay (unsigned int value)   // Delay routine
      {
            unsigned int x,y;
            for (x=0;x<900;x++)
            for (y=0;y<value;y++);
      }
```

```
unsigned char getkey ()             // Routine to read matrix keyboard
      {
            unsigned char colloc, rowloc;
            TMOD = 0x20;
            TH1 = -3;
            SCON = 0x50;
            TR1 = 1;
            P2 = 0xff;
                  do    {
                              P2 = 0x0f;        //Wait untill all keys
                                                //are released
                              colloc = P2;
                              colloc &= 0x0f;
                        } while  (colloc != 0x0f);

            do    {
                  do
                        {
                              MSDelay (1);
                              colloc = P2;
                              colloc &= 0x0f;
                        } while  (colloc == 0x0f);  //Check whether any
                                                    //key is pressed
                        MSDelay (1);
                        colloc = P2;
                        colloc &= 0x0f;     //Confirm whether any
                                           //key is pressed after delay
                        } while  (colloc == 0x0f);    //to aviod spikes

                        while(1)
                        {
                              P2 = 0xfE;        //get the row presses
                              colloc = P2;
                              colloc &= 0xf0;
                              if (colloc != 0xf0)
                                    {
                                          rowloc = 0;
                                          break;
                                    }
                              P2 = 0xfd;
                              colloc = P2;
                              colloc &= 0xf0;
                              if (colloc != 0xf0)
                                    {
                                          rowloc = 1;
                                          break;
                                    }
                              P2 = 0xfb;
                              colloc = P2;
                              colloc &= 0xf0;
                              if (colloc != 0xf0)
```

```
                                    {
                                          rowloc = 2;
                                          break;
                                    }
                              P2 = 0xf7;
                              colloc = P2;
                              colloc &= 0xf0;
                              rowloc = 3;
                              break;
                                    }

                  //get the coloum presses
                  if (colloc == 0xe0) key = (keypad[rowloc][0]);
                  else if (colloc == 0xd0) key = (keypad[rowloc][1]);
                  else if  (colloc == 0xb0) key = (keypad[rowloc][2]);
                  else key = (keypad[rowloc][3]);
                  return(key);
            }
//----------------Hex to Decimal conversion routine

void convert_display(unsigned char value)
      {
            unsigned char x, d1, d2, d3;
            x = value / 10;   //divide by 10
            d1 = value % 10;  //save low digit (reminder of division)
            d2 = x % 10;
            d3 = x / 10;      //divide by 10 once more
            display(d1);
            display(d2);
            display(d3);
            return;
      }

//----------------7 segment display

void display(unsigned char value)
{
      char m;
      char buffer;
      buffer = value;
            for(m=0;m<8;m++)
            {
                  if(buffer&0x80) ser=1;
                  else ser=0;
                  srck=1;
                  buffer<<=1;
                  srck=0;
            }
      rck=1;
      ;
      rck=0; }
```

EXPERIMENT NO.9

ALPHANUMERIC LCD PANEL AND HEX KEYPAD INPUT INTERFACE TO 8051

OBJECTIVE

TO DEMONSTRATE THE BASIC INTERFACE BETWEEN AN LCD DISPLAY AND 4 X 4 MATRIX KEY BOARD. INPUT IS VIA A MATRIX KEYPAD AND OUTPUT IS DISPLAYED ON A 2X16 LCD.

Hardware:

Microcontroller	89s8252
Crystal Freq	11.0592

I/O Port configuration

Port 0	Output	LCD Data
Port 3	Output	LCD Control
Port 2	Input	Matrix Keypad

Keypad:

1	2	3	/
4	5	6	*
7	8	9	-
C	0	=	+

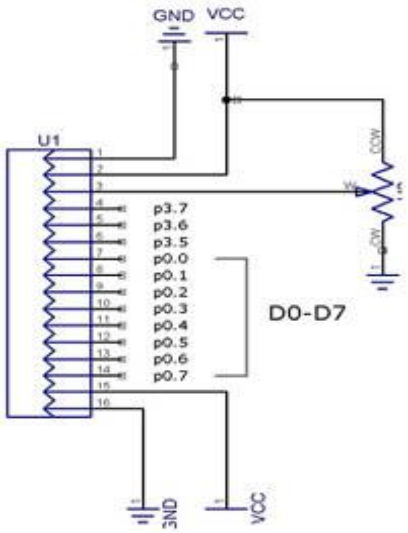


Figure 3. Interface to LCD Display

PROGRAM

```
#include <REG52.H>
      sfr ldata = 0x80; //Variables declaration for LCD
      sbit rs = P3^7;
      sbit rw = P3^6;
      sbit en = P3^5;
      sbit busy = P0^7;
//--------------Funtion declaration
void MSDelay (unsigned int); //Delay
void lcdcmd (unsigned char value);
void lcddata (unsigned char value);
```

```
void lcdready ();
unsigned char getkey ();
unsigned char j,countl,countr, key=0,Data1,Data2,Funct,Ans ;
                                //Serially received data
unsigned char idata msg[13] = {"Key Pressed "};
unsigned char idata keypad[4][4]= { '1','2','3','/',
                                    '4','5','6','*',
                                    '7','8','9','-',
                                    'C','0','C','+',};
void main()
      {
            lcdcmd(0x38);
            lcdcmd(0x0e);
            lcdcmd(0x01);
            lcdcmd(0x06);
            lcdcmd(0x83);
//-------------Display message on lCD terminal
            for (j=0;j<13;j++)
            {
                  lcddata(msg[j]);          //Get data from
                  lookup table
            }

            while(1)
            {
                  lcdcmd(0xC3);
                  getkey (); //get data from matrix key pad
                  lcddata(key);
                  MSDelay (10);
            }
            }
void MSDelay (unsigned int value)            // Delay routine
      {
            unsigned int x,y;
            for (x=0;x<900;x++)
            for (y=0;y<value;y++);
      }
void lcdcmd (unsigned char value)
      {
            lcdready();
            ldata = value;
            rs = 0;
            rw = 0;
            en = 1;
            45Microcontroller 8051
            Sitech Electronics
            MSDelay(1);
            en = 0;
            return;
      }
```

```
void lcddata (unsigned char value)
      {
            lcdready();
            ldata = value;
            rs = 1;
            rw = 0;
            en = 1;
            MSDelay(1);
            en = 0;
            return;
      }

void lcdready ()
      {
            busy = 1;
            rs = 0;
            rw = 1;
            while (busy == 1)
            {
                  en = 0;
                  MSDelay(1);
                  en = 1;
            }
            return;
      }
unsigned char getkey ()              // Routine to read matrix keyboard
      {
      unsigned char colloc, rowloc;
      TMOD = 0x20;
      TH1 = -3;
      SCON = 0x50;
      TR1 = 1;
      P2 = 0xff;
      do {
            P2 = 0x0f;                //Wait untill all keys are released
            colloc = P2;
            colloc &= 0x0f;
          }
      while (colloc != 0x0f);
      do
      {
            do
            {
                  MSDelay (1);
                  colloc = P2;
                  colloc &= 0x0f;
            }
            while (colloc == 0x0f);  //Check whether any key is pressed
            MSDelay (1);
            colloc = P2;
            colloc &= 0x0f;          //Confirm whether any key is
```

```
                                    //pressed after delay
      }
      while (colloc == 0x0f);       //to aviod spikes
      while(1)
      {
            P2 = 0xfE;              //get the row presses
            colloc = P2;
            colloc &= 0xf0;

      if (colloc != 0xf0)
            {
            rowloc = 0;
            break;
            }
            P2 = 0xfd;
            colloc = P2;
            colloc &= 0xf0;
      if (colloc != 0xf0)
            {
            rowloc = 1;
            break;
            }
            P2 = 0xfb;
            colloc = P2;
            colloc &= 0xf0;
      if (colloc != 0xf0)
            {
            rowloc = 2;
            break;
            }
            P2 = 0xf7;
            colloc = P2;
            colloc &= 0xf0;
            rowloc = 3;
            break;
            }
      //get the coloum presses

      if (colloc == 0xe0) key = (keypad[rowloc][0]);

      else if (colloc == 0xd0) key = (keypad[rowloc][1]);

      else if (colloc == 0xb0) key = (keypad[rowloc][2]);

      else key = (keypad[rowloc][3]);

      return(key);
}
```

EXPERIMENT NO.10

EXTERNAL ADC AND TEMPERATURE CONTROL INTERFACE TO 8051

OBJECTIVE

THIS PROGRAM IS IS TO IMPLEMENT A BASIC TEMPERATURE SENSOR USING AN ADC. OUTPUT IS DISPLAYED ON A 2X16 LCD. OUTPUT OF THE SENSOR VARY FROM 27C TO 141C

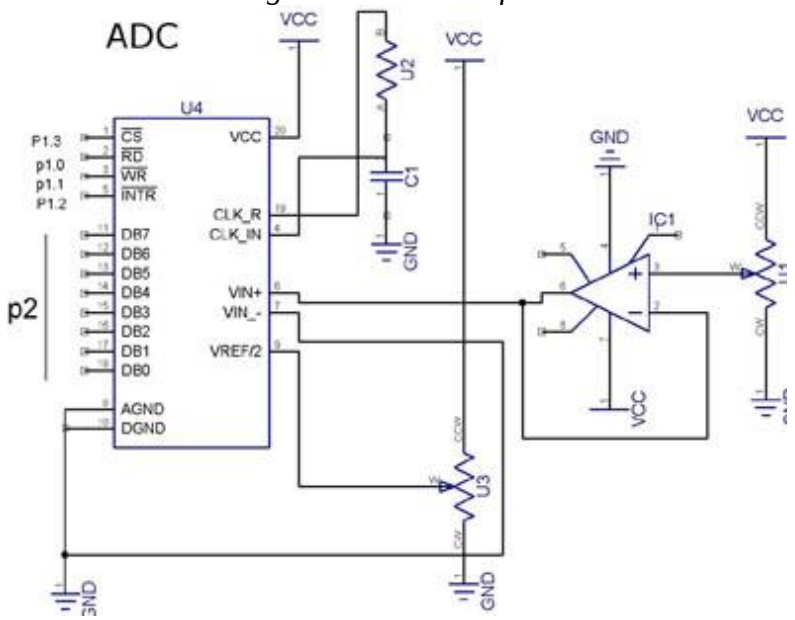
Hardware:

Microcontroller	89s8252	
Crystal Freq	11.0592	

I/O Port configuration

Port 0	output	LCD Data
Port 1	input	Control signal ADC
Port 2	input	ADC
Port	output	LCD Control

Figure 3. ADC Interface



PROGRAM:

```
#include <reg52.h>
sbit rd = P1^1;          //Variables declaration for ADC
sbit wr = P1^2;
sbit intr = P1^3;
sbit aen = P1^0;
sfr mydata1 = 0xA0;      //ADC data
sfr ldata = 0x80;        //Variables declaration for LCD
sbit rs = P3^7;
sbit rw = P3^6;
```

```
      sbit en = P3^5;
      sbit busy = P0^7;
//--------------Funtion declaration
      void convert_display(unsigned char); //Hex to Binary converter
      void lcdcmd (unsigned char value);   //LCD command
      void lcddata (unsigned char value); //LCD data
      void lcdready ();                   //Initialization LCD 2
                                          //lines,5x7 matrix
      void MSDealy(unsigned int itime);   //Delay
void main()
{
      unsigned char value; //ADC data recived
      lcdcmd(0x38); //LCD command
      lcdcmd(0x0e);
      lcdcmd(0x01);
      lcdcmd(0x06);
      lcdcmd(0x83);
      lcddata('T'); //LCD data
      lcddata('e');
      lcddata('m');
      lcddata('p');
      lcddata('e');
      lcddata('r');
      lcddata('a');
      lcddata('t');
      lcddata('u');
      lcddata('r');
      lcddata('e');
      lcddata('r');
      mydata1 = 0xff;                   //Set port 2 as input
      ldata = 0x00;                     //Set port 0 as output
//---------ADC routine
while (1)
      {
            aen = 0;                 //enable ADC
            wr = 0;                  //Write=0
            wr = 1;                 //WR=1 L-TO-H TO START CONVERSION
            while (intr == 1) ;     //WAIT FOR END OF CONVERSION
            rd = 0;     //CONVERSION FINISHED,ENABLE RD
            value = mydata1;        //READ THE DATA
            value = value - 114;
            lcdcmd(0xC7);
            convert_display(value);
            rd = 1;                  //MAKE RD=1 FOR NEXT ROUND
            MSDealy(10);
      }
}
//----------------Hex to Binary convertion routine

void convert_display(unsigned char value)
      {
```

```
            unsigned char x, d1, d2, d3, data1, data2, data3;
            x = value / 10;           //divide by 10
            d1 = value % 10;          //save low digit (reminder of
                                      //division)
            d2 = x % 10;
            d3 = x / 10;             //divide by 10 once more
            data1 = d1 | 0x30;       //make it ASCII and save LSB
            data2 = d2 | 0x30;
            data3 = d3 | 0x30;
            lcddata(data3);         //display converted output MSB
                                    //first
            lcddata(data2);
            lcddata(data1);
            return;
      }
//--------------LCD command
      void lcdcmd (unsigned char value)
      {
            lcdready();               //is LCD ready?
            ldata = value;            //issue command code
            rs = 0;                  //RS=0 for command
            rw = 0;                  //R/W=0 to write to LCD
            en = 1;                  //E=1 for H-to-L pulse
            MSDealy(1);
            en = 0;                  //E=0 ,latch in
            return;
      }
//-------------LCD data
      void lcddata (unsigned char value)
      {
            lcdready();                //is LCD ready?
            ldata = value;             //issue data
            rs = 1;                    //RS=1 for data
            rw = 0;                   //R/W=0 to write to LCD
            en = 1;                   //E=1 for H-to-L pulse
            MSDealy(1);
            en = 0;                   //E=0, latch in
            return;
      }
      void lcdready ()
      {
            busy = 1;                  //make P1.7 input port
            rs = 0;                    //RS=0 access command reg
            rw = 1;                   //R/W=1 read command reg ;read
                                      //command reg and check
            busy flag
            while (busy == 1)         //stay until busy flag=0
            {
                  en = 0;            //E=1 for H-to-L pulse
                  MSDealy(1);
                  en = 1;            //E=0 H-to-L pulse
```

```
            }
            return;
      }
//------------Delay Routine
void MSDealy(unsigned int itime)
      {
            unsigned int i,j;
            for (i=0;i<itime;i++)
            for (j=0;j<1275;j++);
      }
```

EXPERIMENT NO.11

GENERATE DIFFERENT WAVEFORMS SINE, SQUARE, TRIANGULAR, RAMP ETC. USING DAC INTERFACE TO 8051; CHANGE THE FREQUENCY AND AMPLITUDE

OBJECTIVE

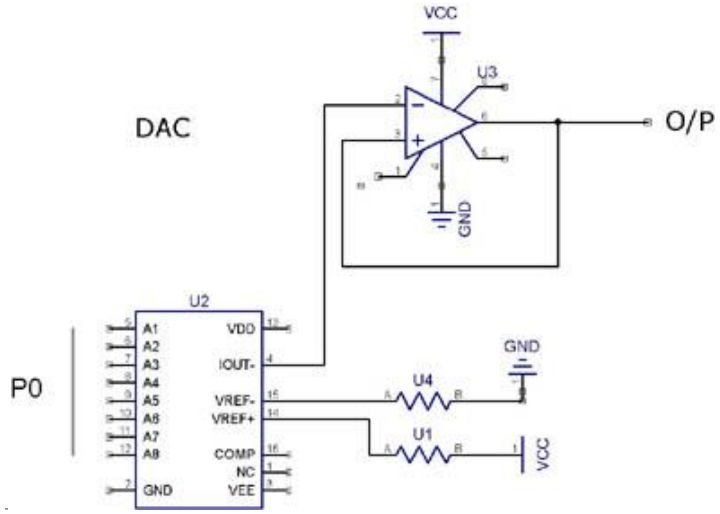
THIS PROGRAM IMPLEMENTS THE BASIC WAVE FORM GENERATION USING DAC. OUTPUT IS DISPLAYED ON A CRO.

Hardware:

Microcontroller	89s8252
Crystal Freq	11.0592

I/O Port configuration

Port output DAC



10.1 SQUARE WAVE GENERATION

```
#include <reg52.h>
idata unsigned int r;
void main()
{
      while(1)
      {
            P0 = 0x00;  for(r=0;r<400;r++);
            P0 = 0xff;  for(r=0;r<400;r++);
      }
}
```

10.2 TRIANGLE WAVE GENERATION

```
#include <reg52.h>
#define DAC_IN P0
void main ()
{
      DAC_IN =0x00;
      while(1)
```

```
      {
            while(DAC_IN < 0xff)    { DAC_IN = DAC_IN + 0x01; }
            while(DAC_IN > 0x00)    { DAC_IN = DAC_IN - 0x01; }
      }
}
```

10.3 STAIRCASE WAVE GENERATION

```
#include <reg52.h>
code unsigned char array[6]={0x00,0x33,0x66,0x99,0xcc,0xff};
void main ()
{
      unsigned char i,r;
      while(1)
      {
            for(i=0;i<5;i++)
            {
                  P0 = array[i];
                  for(r=0;r<40;r++);
            }
      }
}
```

10.4 POSITIVE RAMP WAVE GENERATION

```
#include <reg52.h>
void main ()
{
      while(1)
      {
            P0 =P0 + 0x01;
      }
}
```

10.5 NEGATIVE RAMP WAVE GENERATION

```
#include <reg52.h>
void main ()
{
      while(1)
      {
            P0 =P0 - 0x01;
      }
}
```

10.6 SINE WAVE GENERATION

```
#include <reg52.h>
#include<math.h>
```

```
unsigned char arr[62];
float x;
unsigned int i=0;
void main()
{ P0=0xFF;
     for (x = 0; x < (2 * 3.1415); x += 0.1)
     {
           arr[i]=127+127 * sin(x);
           i++;
     }
     P0=0X00;
     while(1)
{
     for (i=0;i<62;i++)
     {
           P0=arr[i];
     }
}
}
```

EXPERIMENT NO.12

STEPPER AND DC MOTOR CONTROL INTERFACE TO 8051

12.1 STEPPER MOTOR CONTROL

OBJECTIVE

THIS PROGRAM DEMONSTRATES A STEPPER MOTOR CONTROL USING AN H-BRIDGE CONTROLLER. DIRECTION AND SPEED ARE SELECTED WITH HYPER TERMINAL.

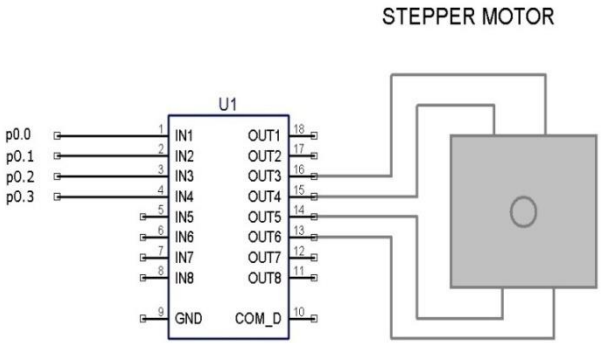
Hardware:

Microcontroller	89s8252
Crystal Freq	11.0592

I/O Port configuration

Port 0	output	DC motor

STEPPER MOTOR MODULE




```
#include <REG52.H>
void MSDelay (unsigned int);
void Send (unsigned char);
unsigned int  rdata=1;        //for clockwise 0 and anticlockwise 1
void main()
      {
           if(rdata==0)
            {
                while (1)     //keep doing the function until next command
is received
                        {
                        P0 =0x66;          //Clockwise Rotation
                        MSDelay (100);     //to change speed change delay
                        P0 =0xCC;
                        MSDelay (100);
                        P0 =0x99;
                        MSDelay (100);
                        P0 =0x33;
```

```
                        MSDelay (100);
                        }
                         }
                  else
                  {
                        while (1)
                        {
                        P0 =0x33;                  //AntiClockwise Rotation
                        MSDelay (100);
                        P0 =0x99;
                        MSDelay (100);
                        P0 =0xCC;
                        MSDelay (100);
                        P0 =0x66;
                        MSDelay (100);
                        }

                        }
      }

void MSDelay (unsigned int value) // Delay rouitne
   {
                     unsigned int x,y;
         for (x=0;x<600;x++)
         for (y=0;y<value;y++);
   }
```

12.2 DC MOTOR CONTROL

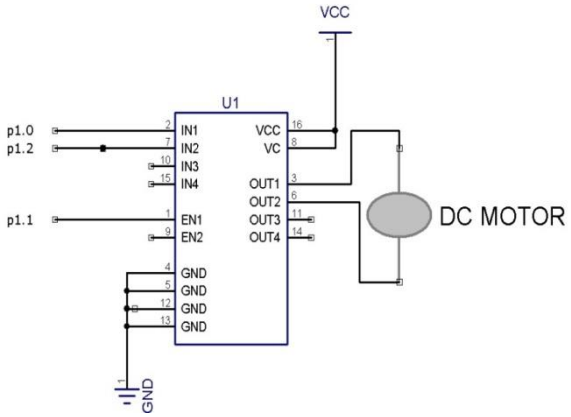
OBJECTIVE
THIS PROGRAM DEMONSTRATES A DC MOTOR CONTROL USING AN H-BRIDGE CONTROLLER. DIRECTION AND SPEED ARE SELECTED WITH HYPER TERMINAL.
Hardware:

Microcontroller	89s8252
Crystal Freq	11.0592

I/O Port configuration

Port 0	output ;DC motor

DC MOTOR




```
#include<reg52.h>
void msdelay(unsigned int);
sbit enable= P0^6;
sbit mtr_1=P0^7;
sbit mtr_2=P0^4;
unsigned int rot=1;                    //for clkwise 0 and anti clkwise 1
void main()
{
     if(rot==0)
     {
           while(1)
                       {
                    msdelay(1000);
                   mtr_1=1;
                   mtr_2=0;
                   msdelay(1000);
                    }
    }
  else
        {
      while(1)
                       {
                       msdelay(1000);
                        mtr_1=0;
                       mtr_2=1;
                       msdelay(1000);
                          }
           }
  }

void msdelay(unsigned int value)
  {
  unsigned int x,y;
  for(x=0;x<500;x++)
  for(y=0;y<value;y++);
  }
```

EXPERIMENT NO.13

ELEVATOR INTERFACE TO 8051

OBJECTIVE

THIS PROGRAM IS CAPABLE OF DEMONSTRATING THE BASIC WORKING OF ELEVATOR . INPUT IS VIA A MATRIX KEYPAD AND OUTPUT IS DISPLAYED ON A 2X16 LCD.

Hardware:

Microcontroller	89s8252
Crystal Freq	11.0592

I/O Port configuration

Port 0	output; LCD Data
Port 1	NA; Open
Port 2	input; Matrix Keypad
Port 3	output; LCD Control

Keypad:

1 2 3
4 5 6
7 8 9
0 ; 0 ----> up/down

```
#include <REG52.H>

sfr ldata = 0x80;                          //Variables declaration for LCD
sbit rs = P3^7;
sbit rw = P3^6;
sbit en = P3^5;
sbit busy = P0^7;
sbit carry = PSW^7;

//--------------Funtion declaration

void convert_display(unsigned char);       //Hex to Binary converter
void MSDelay (unsigned int);               //Delay
void lcdcmd (unsigned char value);
void lcddata (unsigned char value);
void lcdready ();
void down ();
void up ();
unsigned char getkey ();
unsigned char i,j,countl,countr, key=0, Data1='0',
                    Data2='0', Funct, Ans ;
                                    //Serially recieved data
                                    //-----------------The message
            unsigned char idata msg[9] =  {"Floor No:"};
```

```
            unsigned char idata keypad[4][4]= { '1','2','3','X',
                                                '4','5','6','X',
                                                '7','8','9','X',
                                                'X','0','X','X',};


void main()
      {
            lcdcmd(0x38);
            lcdcmd(0x0e);
            lcdcmd(0x01);
            lcdcmd(0x06);
            lcdcmd(0x80);
                  for (j=0;j<9;j++)
                     {
                        lcddata(msg[j]);      //Get data from lookup
                                              //table
                      }
                        lcdcmd(0x8A);
                        lcddata('0');
                        lcddata('0');
                  while(1)
                        {
                        do
                        {
                        getkey ();       //get data from matrix key pad
                        MSDelay(10);
                        Data1 = key;
                        key = key & 0xF0;
                        }
                        while (key!=0x30);

                        carry =0;
                        if (Data1<Data2) down();
                        else up();
                        }
          }
```

```
                     void down()
```

```
                        {
                        for(i=(Data2-Data1);i>0;i--)
                              {
                              for (j=3;j>0;j--)
                                    {
                                    lcdcmd(0x8E);
                                    lcddata('v');
                                    lcddata('v');
                                    MSDelay(30);
                                    lcdcmd(0x8E);
                                    lcddata(' ');
                                    lcddata(' ');
                                    MSDelay(30);
                                    }

                              Data2--;
                              lcdcmd(0x8b);
                              lcddata(Data2);

                              }
                        }

                  void up()
                        {
                        for(i=(Data1-Data2);i>0;i--)
                              {
                              for (j=3;j>0;j--)
                                    {
                                    lcdcmd(0x8E);
                                    lcddata('^');
                                    lcddata('^');
                                    MSDelay(30);
                                    lcdcmd(0x8E);
                                    lcddata(' ');
                                    lcddata(' ');
                                    MSDelay(30);
                                    }

                              Data2++;
                              lcdcmd(0x8b);
                              lcddata(Data2);

                              }
                        }


void MSDelay (unsigned int value)             // Delay routine
      {
            unsigned int x,y;
            for (x=0;x<900;x++)
            for (y=0;y<value;y++);
```

```
	}


void lcdcmd (unsigned char value)
	{
		lcdready();
		ldata = value;
		rs = 0;
		rw = 0;
		en = 1;
		MSDelay(1);
		en = 0;
		return;
	}

void lcddata (unsigned char value)
	{
		lcdready();
		ldata = value;
		rs = 1;
		rw = 0;
		en = 1;
		MSDelay(1);
		en = 0;
		return;
	}
void lcdready ()
	{
		busy = 1;
		rs = 0;
		rw = 1;
		while (busy == 1)
			{
				en = 0;
				MSDelay(1);
				en = 1;
			}
		return;
	}

unsigned char getkey ()                    // Routine to read matrix
keyboard
	{
		unsigned char colloc, rowloc;
		TMOD = 0x20;
		TH1 = -3;
		SCON = 0x50;
		TR1 = 1;

		P2 = 0xff;
```

```
                    do    {
                          P2 = 0x0f;  //Wait untill all keys are released
                                colloc = P2;
                                colloc &= 0x0f;
                          } while  (colloc != 0x0f);




do    {
                        do
                          {
                             MSDelay (1);
                             colloc = P2;
                             colloc &= 0x0f;
                          } while  (colloc == 0x0f);
                                     //Check whether any key is pressed
                          MSDelay (1);
                          colloc = P2;
                          colloc &= 0x0f;
                    //Confirm whether any ket is pressed after delay
                          } while  (colloc == 0x0f);//to aviod spikes

                          while(1)
                          {
                                P2 = 0xfE;        //get the row presses
                                colloc = P2;
                                colloc &= 0xf0;
                                if (colloc != 0xf0)
                                      {
                                            rowloc = 0;
                                            break;
                                      }
                                P2 = 0xfd;
                                colloc = P2;
                                colloc &= 0xf0;
                                if (colloc != 0xf0)
                                      {
                                            rowloc = 1;
                                            break;
                                      }
                                P2 = 0xfb;
                                colloc = P2;
                                colloc &= 0xf0;
                                if (colloc != 0xf0)
                                      {
                                            rowloc = 2;
                                            break;
                                      }
                                P2 = 0xf7;
```

```
                colloc = P2;
                colloc &= 0xf0;
                rowloc = 3;
                break;

            }               //get the coloum presses

      if (colloc == 0xe0) key = (keypad[rowloc][0]);
      else if (colloc == 0xd0) key = (keypad[rowloc][1]);
      else if  (colloc == 0xb0) key = (keypad[rowloc][2]);
      else key = (keypad[rowloc][3]);     return(key);
}
```

APPENDIX A

GENERAL QUESTIONS

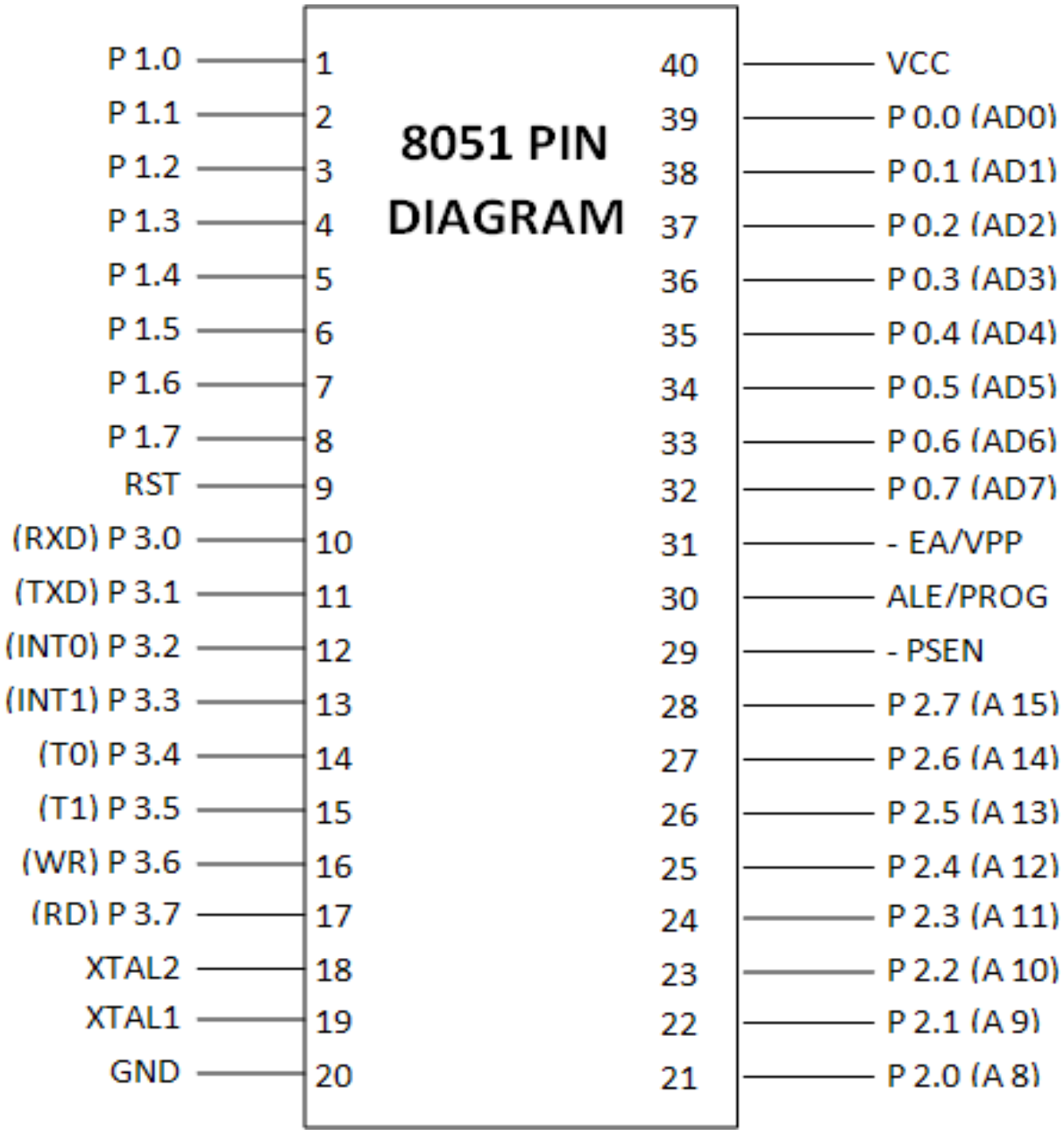
1. Upon reset, all ports of the 8051 are configured as _____ (output, input).
2. Which ports of the 8051 have internal pull-up resistors?
3. Which ports of the 8051 require the connection of external pull-up resistors in order to be used for I/O? Show the drawing for the connection.
4. In the 8051, explain why we must write "1" to a port in order for it to be used for input.
5. Explain why we need to buffer the switches used as input in order to avoid damaging the 8051 port.
6. How does the LCD distinguish data from instruction codes when receiving information at its data pin?
7. To send the instruction code 01 to clear the display, we must make RS = ___.
8. To send letter 'A' to be displayed on the LCD, we must make RS = ____.
9. What is the purpose of the E line? Is it an input or an output as far as the LCD is concerned?
10. When is the information (code or data) on the LCD pin latched into the LCD?
11. Indicate the direction of pins WR, RD, and INTR from the point of view of the 8051.
12. Give the three steps for converting data and getting the data out of the ADC804. State the status of the CS, RD, INTR, and WR pins in each step.
13. Assume that $V_{ref}/2$ is connected to 1.28 V. Find the following.
    13.1. step size
    13.2. maximum range for V_{in}
    13.3. D7 - D0 values if V_{in} = 1.2 V
    13.4. V_{in} if D7 - D0 = 11111111
    13.5. V_{in} if D7 - D0 = 10011100
14. Assume that $V_{ref}/2$ is connected to 1.9 V. Find the following.
    14.1. step size
    14.2. maximum range for V_{in}
    14.3. D7 - D0 values if V_{in} = 2.7 V
    14.4. V_{in} if D7 - D0 = 11111111
    14.5. V_{in} if D7 - D0 = 11011101
15. The ADC804 is a(n) ____-bit converter.
16. To get step size of 2 mV, what is the value for Vref/2?
17. What is a transducer?
18. What is the form of the transducer output?
19. What is preprocessing of transducer signals to be fed into an ADC called?
20. The LM35 and LM34 produce a _____ mV output for every degree of change in temperature.

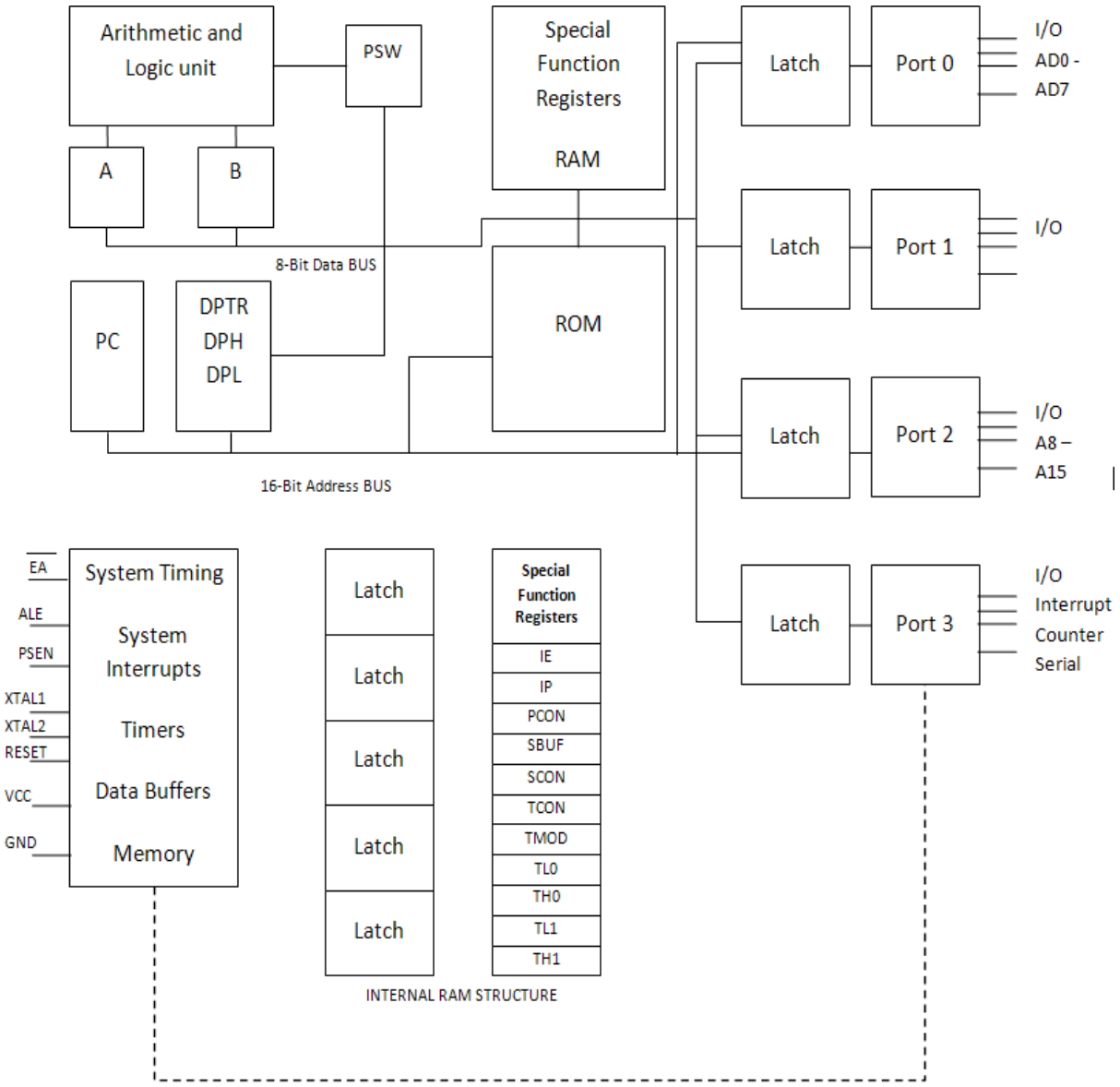
21. The LM35/LM34 is a _____ (linear, nonlinear) device. Discuss the advantages of linear devices and of nonlinear devices.
22. Explain signal conditioning and its role in data acquisition.
23. What is the maximum frequency that can be generated using Mode 1 if the crystal frequency is 11.0592 MHz? Show your calculation.
24. What is the maximum frequency that can be generated using Mode 2 if the crystal frequency is 11.0592 MHz? Show your calculation.
25. What is the lowest frequency that can be generated using Mode 1 if the crystal frequency is 11.0592 MHz? Show your calculation.
26. What is the lowest frequency that can be generated using Mode 1 if the crystal frequency is 11.0592 MHz? Show your calculation.
27. In mode 1, when is TFx set to high?
28. In mode 2, when is TFx set to high?
29. The 8051 TxD and RxD signals _____ (are, are not) TTL-compatible.
30. In this lab, what is the role of the MAX233 (MAX232) chip?
31. With XTAL=11.0592 MHz, what is the maximum baud rate for the 8051?
32. Show how to achieve the maximum baud rate
33. What is the role of TI and RI?
34. True or false. The 8051 can transfer data in full-duplex.
35. For full duplex, what are the absolute minimum signals needed between the 8051 and the PC? Give their names.
36. What is a step angle? Define steps per revolution.
37. If a given stepper motor has a step angle of 5 degrees, find the number of steps per revolution.
38. Give the four sequences for counter clockwise if it starts with 10011001 (binary).
39. Using the "RL A" instruction, show the four-step sequences if the initial step is 0011 (binary).
40. Give the number of times the four-step sequence must be applied to a stepper motor to make a 100-degree move if the motor has a 5-degree step angle. Also fill in the characteristics for your motor below.
    40.1. Step angle _____Degree of movement per 4-step sequence _____
    40.2. Steps per revolution _____ Number of rotor teeth _____
    40.3. What is the purpose of generating the truth table for a given keyboard?
41. What is the purpose of grounding each row in keyboard interfacing?
42. What is the input to the microcontroller from column if no key is pressed?
43. True or false. In our N x M matrix keypad program we cannot press two keys at the same time.
44. In your program in, how is the key press detected?
45. In your program in, how is a key press identified?
46. Explain the role of the C/T bit in the TMOD register.
47. How is the 8051 used as an event counter to count an external event?
48. If timer/counter 0 is used as an event counter, what is the maximum count for the following modes.
    48.1. Mode 1 48.2 Mode 2

49. Indicate which pin is used for the following.
    49.1. timer/counter 0 49.2 timer/counter 1
50. If timer/counter 0 is used in mode 1 to count an external event, explain when TF0 is set to high.
51. If timer/counter 1 is used in mode 2 to count an external event, explain when TF0 is set to high.
52. Indicate the direction of pins ALE, SC, EOC, and OE from the point of view of the ADC808/809.
53. Give the steps for converting data and getting the data out of the ADC809. State the status of the SC and EOC pins in each step.
54. Give the role of signals ALE, A, B, and C in selecting the ADC channel.
55. In the ADC809 assume that V_{ref} is connected to 2.56 V. Find the following.
    55.1. step size
    55.2. maximum range for V_{in}
    55.3. D7 - D0 values if V_{in} = 1.2 V
    55.4. V_{in} if D7 - D0 = 11111111
    55.5. V_{in} if D7 - D0 = 10011100
56. In the ADC809 assume that V_{ref} is connected to 5V. Find the following.
    56.1. step size
    56.2. maximum range for V_{in}
    56.3. D7 - D0 values if Vin = 2.7 V
    56.4. V_{in} if D7 - D0 = 11111111
    56.5. V_{in} if D7 – D0 = 11011101
57. In connecting ADC808/809 to an 8051, indicate the direction of pins ALE, SC, EOC, and OE from the point of view of the 8051.
58. Define the following terminology in DAC.
    58.1. resolution
    58.2. full-scale voltage output
    58.3. settling time
59. For your circuit, find V_{out} for the following inputs.
    59.1. 11001100
    59.2. 10001111
60. To get a smaller step size, we need DAC with _____ (more, less) data bit inputs.
61. In Figure 13-7 of the textbook, assume that R = 2.5 K ohms. Calculate V_{out} for the following binary inputs.
    61.1. 11000010
    61.2. 01000001
    61.3. 00101100
    61.4. 11111111
62. Name all of the interrupts in the 8051 and their vector table addresses.
63. In timer mode 1, indicate when TF0 causes the interrupt.
64. In timer mode 2, indicate when TF0 causes the interrupt.
65. On reset, INT0 (and INT1) are _____ (edge, level) triggered.
66. On reset, which interrupt has the highest priority?
67. True or False. There is only a single interrupt for the serial data transfer.

APPENDIX B

8051 PIN DIAGRAM AND ARCHITECTURE




Arithmetic and Logic unit
PSW
Special Function Registers
RAM
Latch
Port 0
I/O
AD0 - AD7
A
B
Latch
Port 1
I/O
8-Bit Data BUS
PC
DPTR
DPH
DPL
ROM
Latch
Port 2
I/O
A8 – A15
16-Bit Address BUS
EA
ALE
PSEN
XTAL1
XTAL2
RESET
VCC
GND
System Timing
System Interrupts
Timers
Data Buffers
Memory
Latch
Latch
Latch
Latch
Latch
Special Function Registers
IE
IP
PCON
SBUF
SCON
TCON
TMOD
TL0
TH0
TL1
TH1
Latch
Port 3
I/O
Interrupt
Counter
Serial
INTERNAL RAM STRUCTURE

APPENDIX C

INSTRUCTION SET SUMMARY

Instruction Set Summary

Mnemonic		Description	Byte	Cycle
Arithmetic Operations				
ADD	A,Rn	Add register to accumulator	1	1
ADD	A,direct	Add direct byte to accumulator	2	1
ADD	A, @Ri	Add indirect RAM to accumulator	1	1
ADD	A,#data	Add immediate data to accumulator	2	1
ADDC	A,Rn	Add register to accumulator with carry flag	1	1
ADDC	A,direct	Add direct byte to A with carry flag	2	1
ADDC	A, @Ri	Add indirect RAM to A with carry flag	1	1
ADDC	A, #data	Add immediate data to A with carry flag	2	1
SUBB	A,Rn	Subtract register from A with borrow	1	1
SUBB	A,direct	Subtract direct byte from A with borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with borrow	1	1
SUBB	A,#data	Subtract immediate data from A with borrow	2	1
INC	A	Increment accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
DEC	A	Decrement accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
INC	DPTR	Increment data pointer	1	2
MUL	AB	Multiply A and B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal adjust accumulator	1	1

Instruction Set Summary (cont'd)

Mnemonic		Description	Byte	Cycle
Logic Operations				
ANL	A,Rn	AND register to accumulator	1	1
ANL	A,direct	AND direct byte to accumulator	2	1
ANL	A,@Ri	AND indirect RAM to accumulator	1	1
ANL	A,#data	AND immediate data to accumulator	2	1
ANL	direct,A	AND accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to accumulator	1	1
ORL	A,direct	OR direct byte to accumulator	2	1
ORL	A,@Ri	OR indirect RAM to accumulator	1	1
ORL	A,#data	OR immediate data to accumulator	2	1
ORL	direct,A	OR accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive OR register to accumulator	1	1
XRL	A direct	Exclusive OR direct byte to accumulator	2	1
XRL	A,@Ri	Exclusive OR indirect RAM to accumulator	1	1
XRL	A,#data	Exclusive OR immediate data to accumulator	2	1
XRL	direct,A	Exclusive OR accumulator to direct byte	2	1
XRL	direct,#data	Exclusive OR immediate data to direct byte	3	2
CLR	A	Clear accumulator	1	1
CPL	A	Complement accumulator	1	1
RL	A	Rotate accumulator left	1	1
RLC	A	Rotate accumulator left through carry	1	1
RR	A	Rotate accumulator right	1	1
RRC	A	Rotate accumulator right through carry	1	1
SWAP	A	Swap nibbles within the accumulator	1	1

Instruction Set Summary (cont'd)

Mnemonic		Description	Byte	Cycle
Data Transfer				
MOV	A,Rn	Move register to accumulator	1	1
MOV	A,direct *)	Move direct byte to accumulator	2	1
MOV	A,@Ri	Move indirect RAM to accumulator	1	1
MOV	A,#data	Move immediate data to accumulator	2	1
MOV	Rn,A	Move accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct byte	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri, #data	Move immediate data to indirect RAM	2	1
MOV	DPTR, #data16	Load data pointer with a 16-bit constant	3	2
MOVC	A,@A + DPTR	Move code byte relative to DPTR to accumulator	1	2
MOVC	A,@A + PC	Move code byte relative to PC to accumulator	1	2
MOVX	A,@Ri	Move external RAM (8-bit addr.) to A	1	2
MOVX	A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVX	@Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVX	@DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with accumulator	1	1
XCH	A,direct	Exchange direct byte with accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with accumulator	1	1
XCHD	A,@Ri	Exchange low-order nibble indir. RAM with A	1	1

*) MOV A,ACC is not a valid instruction

Instruction Set Summary (cont'd)

Mnemonic		Description	Byte	Cycle
Boolean Variable Manipulation				
CLR	C	Clear carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set carry flag	1	1
SETB	bit	Set direct bit	2	1
CPL	C	Complement carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to carry flag	2	2
ANL	C,/bit	AND complement of direct bit to carry	2	2
ORL	C,bit	OR direct bit to carry flag	2	2
ORL	C,/bit	OR complement of direct bit to carry	2	2
MOV	C,bit	Move direct bit to carry flag	2	1
MOV	bit,C	Move carry flag to direct bit	2	2
Program and Machine Control				
ACALL	addr11	Absolute subroutine call	2	2
LCALL	addr16	Long subroutine call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute jump	2	2
LJMP	addr16	Long iump	3	2
SJMP	rel	Short jump (relative addr.)	2	2
JMP	@A + DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if accumulator is zero	2	2
JNZ	rel	Jump if accumulator is not zero	2	2
JC	rel	Jump if carry flag is set	2	2
JNC	rel	Jump if carry flag is not set	2	2
JB	bit,rel	Jump if direct bit is set	3	2
JNB	bit,rel	Jump if direct bit is not set	3	2
JBC	bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE	A,direct,rel	Compare direct byte to A and jump if not equal	3	2

Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle

Program and Machine Control (cont'd)

CJNE	A,#data,rel	Compare immediate to A and jump if not equal	3	2
CJNE	Rn,#data rel	Compare immed. to reg. and jump if not equal	3	2
CJNE	@Ri,#data,rel	Compare immed. to ind. and jump if not equal	3	2
DJNZ	Rn,rel	Decrement register and jump if not zero	2	2
DJNZ	direct,rel	Decrement direct byte and jump if not zero	3	2
NOP		No operation	1	1

Notes on Data Addressing Modes

Rn - Working register R0-R7

direct - 128 internal RAM locations, any I/O port, control or status register

@Ri - Indirect internal or external RAM location addressed by register R0 or R1

#data - 8-bit constant included in instruction

#data 16 - 16-bit constant included as bytes 2 and 3 of instruction

bit - 128 software flags, any bitaddressable I/O pin, control or status bit

A - Accumulator

Notes on Program Addressing Modes

addr16 - Destination address for LCALL and LJMP may be anywhere within the 64-Kbyte program memory address space.

addr11 - Destination address for ACALL and AJMP will be within the same 2-Kbyte page of program memory as the first byte of the following instruction.

rel - SJMP and all conditional jumps include an 8 bit offset byte. Range is + 127/– 128 bytes relative to the first byte of the following instruction.