Microcontroller System Laboratory

Experiment 5

Utkarsh Patel 18EC35034

Part 1 – Displaying DOB on LCD via UART Tx-Rx

Objective

Using the keypad, enter your date of birth (DD/MM/YYYY format), transmit it on serial port, receive it on serial port and display on the LCD.

Circuit Diagram

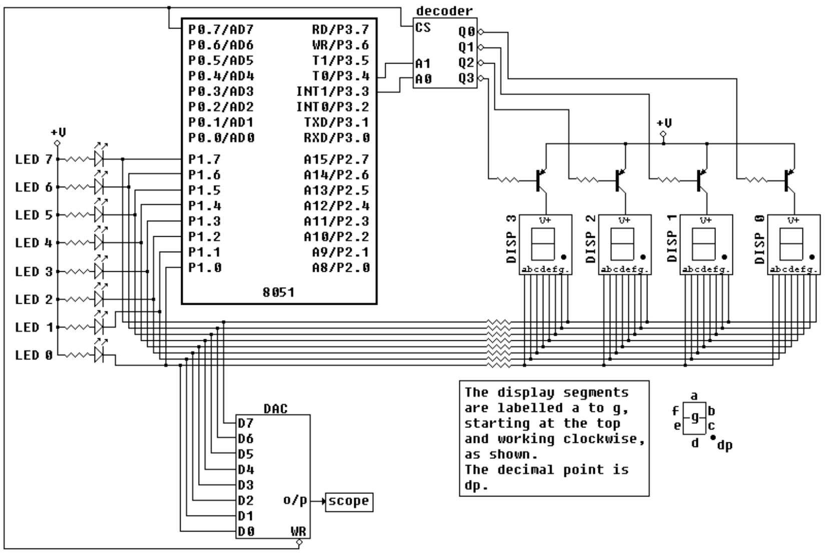

Fig. 1. Circuit diagram

Code

```
; Run this code with update frequency 1000

startReading:
  MOV R7, #0          ; number of keys read till now

loopRead:
  CJNE R7, #8, scanKey   ; reading 8 characters (DD/MM/YYYY)
  JMP configTransmission

scanKey:
  MOV R0, #0          ; | R0 stores the key number being checked
                      ; | and will be locked when a pressed key is detected

  CLR F0;             ; reset flag to indicate scanning is initiated

  SETB P0.3           ; row3 has been checked
  CLR P0.0            ; start checking row0
```

```
  CALL colscan        ; checking columns
  JB F0, keyfound     ; key press identified

  SETB P0.0           ; row0 has been checked
  CLR P0.1            ; start checking row1
  CALL colscan        ; checking columns
  JB F0, keyfound     ; key press identified

  SETB P0.1           ; row1 has been checked
  CLR P0.2            ; start checking row2
  CALL colscan        ; checking columns
  JB F0, keyfound     ; key press identified

  SETB P0.2           ; row2 has been checked
  CLR P0.3            ; start checking row3
  CALL colscan        ; checking columns
  JB F0, keyfound      ; key press identified

  JMP loopRead

colscan:
  JB P0.4, colscan2
  SETB F0
  JNB P0.4, $
  RET

colscan2:
  INC R0
  JB P0.5, colscan3
  SETB F0
  JNB P0.5, $
  RET

colscan3:
  INC R0
  JB P0.6, colnotfound
  SETB F0
  JNB P0.6, $
  RET

colnotfound:
  INC R0
  RET


keyfound:
  ; the keys identified from keypad are stored in RAM starting from 30H to 37H
  MOV A, #30H
```

```
  ADD A, R7            ; A contains address to store current key
  MOV R1, A            ; copying the address to R1
  MOV A, R0            ; copying identified key to A
  MOV @R1, A           ; writing the key to corresponding address
  INC R7               ; increment key counter
  JMP loopRead

configTransmission:
  ; trasmitting the stored keys via UART

  ; dictionary mapping for keys
  MOV 38H, #0          ; | as DOB is 8 char long (stored in 30H to 37H)
                       ; | make 38H null-terminated

  MOV 40H, #'#'        ; key value for R0 = 0x00
  MOV 41H, #'0'        ; key value for R0 = 0x01
  MOV 42H, #'*'        ; key value for R0 = 0x02
  MOV 43H, #'9'        ; key value for R0 = 0x03
  MOV 44H, #'8'        ; key value for R0 = 0x04
  MOV 45H, #'7'        ; key value for R0 = 0x05
  MOV 46H, #'6'        ; key value for R0 = 0x06
  MOV 47H, #'5'        ; key value for R0 = 0x07
  MOV 48H, #'4'        ; key value for R0 = 0x08
  MOV 49H, #'3'        ; key value for R0 = 0x09
  MOV 4AH, #'2'        ; key value for R0 = 0x0A
  MOV 4BH, #'1'        ; key value for R0 = 0x0B

  ; configuring for transmission
  CLR SM0              ; |
  SETB SM1             ; | put serial port in 8-bit UART mode

  MOV A, PCON          ; |
  SETB ACC.7           ; |
  MOV PCON, A          ; | set SMOD in PCON to double baud rate

  MOV TMOD, #20H       ; put timer 1 in 8-bit auto-reload interval timing mode
  MOV TH1, #243        ; put -13 in timer 1 high byte (timer will overflow every 13 us)
  MOV TL1, #243        ; put same value in low byte so when timer is first started it will
overflow after 13 us
  SETB TR1             ; start timer 1

  MOV R0, #0           ; number of characters transmitted till now

initTransmission:
  CJNE R0, #8, loopTransmission  ; transmitting 8 characters
  JMP configReceiver             ; if 8 characters are transmitted, configure receiver

loopTransmission:
```

```
  MOV A, #30H             ; |
  ADD A, R0               ; | correct address to read current key
  MOV R1, A               ; |
  MOV A, @R1              ; |
  ADD A, #40H             ; |
  MOV R1, A               ; |
  MOV A, @R1;             ; | decoding key value from key index

  JZ configReceiver       ; if A contains null value, stop transmission and configure
receiver
  MOV SBUF, A             ; move data to be sent to the serial port
  INC R0                  ; increment R0 to point at next byte of data to be sent
  JNB TI, $               ; wait for TI to be set, indicating serial port has finished
sending byte
  CLR TI                  ; clear TI
  JMP initTransmission

configReceiver:
  SETB REN                ; enable serial port receiver
  MOV R0, #50H            ; characters received from serial port will be stored in RAM
starting with address 50H

initReceiver:
  JNB RI, $               ; wait for byte to be received
  CLR RI                  ; clear the RI flag
  MOV A, SBUF             ; move received byte to A
  CJNE A, #0DH, loopReceiver ; compare it with 0DH - it it's not, skip next instruction
  JMP initLCD             ; if it is the terminating character, jump to the LCD module

loopReceiver:
  MOV @R0,A               ; move from A to location pointed to by R0
  INC R0                  ; increment R0 to point at next location where data will be
stored
  JMP initReceiver

initLCD:
  call configureFor4BitOperation
  call incrementCursorMode
  call displayOnCursonOnBlinkingOn
  call main


configureFor4BitOperation:
  ; for configuring 4-bit operation in LCD

  clr P1.3     ; instruction flow mode

  clr P1.7     ; |
```

```
  clr P1.6     ; |
  setb P1.5    ; |
  clr P1.4     ; | high nibble set

  setb P1.2    ; |
  clr P1.2     ; | negative edge

  call delay

  setb P1.2    ; |
  clr P1.2     ; | negative edge

  setb P1.7

  setb P1.2    ; |
  clr P1.2     ; | negative edge

  call delay

  ret

incrementCursorMode:
  ; for displaying next character on adjacent display
  ; Code = 0x06 = 0000 0110

  clr P1.3   ; instruction mode on

  clr P1.7   ; |
  clr P1.6   ; |
  clr P1.5   ; |
  clr P1.4   ; | high nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  setb P1.6  ; |
  setb P1.5  ; | low nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  call delay

  ret

displayOnCursonOnBlinkingOn:
  ; turning on the display and cursor and choosing blinking
  ; Code = 0x0F = 0000 1111
```

```
  clr P1.3   ; instruction mode on

  clr P1.7   ; |
  clr P1.6   ; |
  clr P1.5   ; |
  clr P1.4   ; | high nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  setb P1.7  ; |
  setb P1.6  ; |
  setb P1.5  ; |
  setb P1.4  ; | low nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  call delay

  ret


main:
  SETB P1.3     ; clear RS - indicates that data is being sent to module
  MOV R1, #50H  ; data to be sent to LCD is stored in 8051 RAM, starting at location 30H
  acall loop

loop:
  MOV A, @R1          ; move data pointed to by R1 to A
  JZ finish           ; if A is 0, then end of data has been reached - jump out of loop
  CALL sendCharacter  ; send data in A to LCD module
  INC R1              ; point to next piece of data
  JMP loop            ; repeat

finish:
  JMP $

sendCharacter:
  MOV C, ACC.7    ; |
  MOV P1.7, C     ; |
  MOV C, ACC.6    ; |
  MOV P1.6, C     ; |
  MOV C, ACC.5    ; |
  MOV P1.5, C     ; |
  MOV C, ACC.4    ; |
  MOV P1.4, C     ; | high nibble set
```

```
  SETB P1.2       ; |
  CLR P1.2        ; | negative edge on E

  MOV C, ACC.3    ; |
  MOV P1.7, C     ; |
  MOV C, ACC.2    ; |
  MOV P1.6, C     ; |
  MOV C, ACC.1    ; |
  MOV P1.5, C     ; |
  MOV C, ACC.0    ; |
  MOV P1.4, C     ; | low nibble set

  SETB P1.2       ; |
  CLR P1.2        ; | negative edge on E

  CALL delay      ; wait for BF to clear

delay:
  MOV R0, #50
  DJNZ R0, $
  RET
```

Discussion

1 LCD Module

- The LCD module consists of 16 rows and 2 columns of 5x8 dot matrices.
- Name of the pins and their corresponding functions are as follows:

Pin Name	Function
VSS	Must be grounded
VCC	5V DC power supply
RS	Register Selection
R/W	Read/write
E	Enable
DB[7:0]	Data

- From the circuit diagram, it can be observed that P1.3 is the RS of the LCD display, and P1.2 is the E of the LCD display.
- To execute any set of command, a negative edge has to be generated by E, i.e., set P1.2 to logic high and then to logic low and add some delay
- There are two types of register modes:
  - **Command mode**: Indicates flow of instruction to the LCD module, RS (P1.3) must be set to logic low to select this register mode
  - **Data mode**: Indicates flow of data to the LCD module, RS (P1.3) must be set to logic high to select this register mode
- Every operation linked with LCD display has a unique hexadecimal code. Some of them are:

Code (in hexadecimal)	Operation
0F	LCD ON, cursor ON, blinking ON
01	Clear screen
02	Return home
04	Decrement cursor
06	Increment cursor
0E	Display ON, cursor OFF
80	Force cursor to the beginning of 1st line
C0	Force cursor to the beginning of 2nd line
38	Use 2 lines and 5x7 matrix
83	Cursor line 1 position 3
3C	Activate second line
08	Display OFF, cursor OFF
C1	Jump to second line, position 1
C2	Jump to second line, position 2
0C	Display ON, cursor OFF

- To perform any operation with code, say 0x75 = 01110101B, we divide the instruction into high nibble set (0111 in this case) and low nibble set (0101 in this case). Then we do:

```
CLR P1.7     ;  0|
SETB P1.6    ;  1|
SETB P1.5    ;  1|
SETB P1.4    ;  1| high nibble set
SETB P1.2    ;   |
CLR P1.2     ;   | negative edge on E
CLR P1.7     ;  0|
SETB P1.6    ;  1|
CLR P1.5     ;  0|
SETB P1.4    ;  1| low nibble set
SETB P1.2    ;   |
```

```
CLR P1.2     ;    | negative edge on E
CALL delay
```

- The characters of my name are stored in RAM in from 0x30 to 0x37
- sendCharacter module is used for displaying the current character on the LCD display

2 Scanning Keypad

- While no key is pressed the program scans row0, row1, row2, row3 and back to row0, continuously.
- When a key is pressed the key number is placed in R0.
- For this program, the keys are numbered as:

```
+----+----+----+
| 11 | 10 |  9 |   row3
+----+----+----+
|  8 |  7 |  6 |   row2
+----+----|----+
|  5 |  4 |  3 |   row1
+----+----+----+
|  2 |  1 |  0 |   row0
+----+----+----+
 col2 col1 col0
```

- The pressed key number will be stored in R0. Therefore, R0 is initially cleared.
- Each key is scanned, and if it is not pressed R0 is incremented. In that way, when the pressed key is found, R0 will contain the key's number.
- The general-purpose flag, F0, is used by the column-scan subroutine to indicate whether or not a pressed key was found in that column. If, after returning from colScan, F0 is set, this means the key was found.
- The data received from keypad is stored in RAM from address 30H to 37H, 38H is used for null-termination.

3 UART Transmitter / Receiver

- The data stored in RAM is transmitted via serial port. For this, the serial port is put to 8-bit UART mode. The data is transmitted at 4800 baud rate. To generate this baud rate, timer 1 must overflow every 13 us with SMOD equal to 1
- Then the data is transmitted and address is incremented till null termination is achieved.
- While receiving on serial port, the data is stored on RAM from address 50H to 57H, 58H being used for null-termination.
- These addresses are used to initiate LCD module.

Part 2 – Displaying sum on LCD module via UART Tx-Rx

Objective

Compute the sum of two decimal numbers (1-12) input from the keyboard, transmit sum on serial port, and display the sum on the LCD after receiving it on serial port.

Circuit Diagram

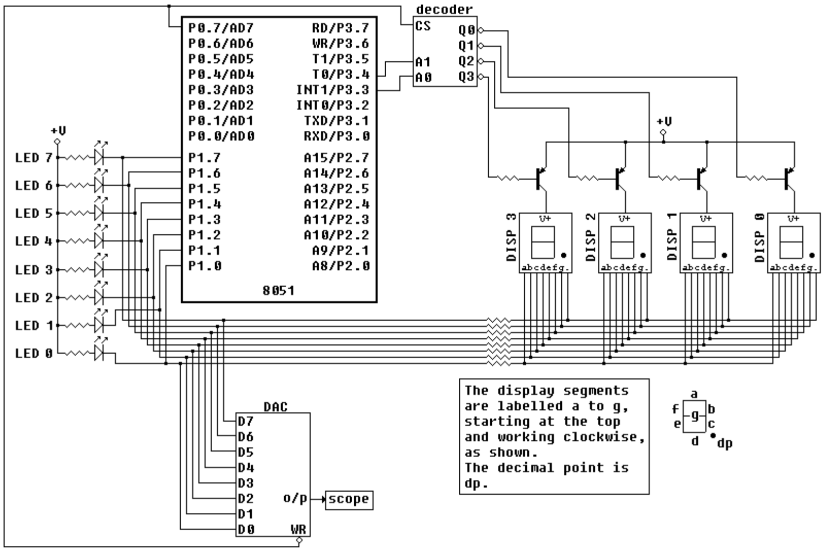

Fig. 2. Circuit diagram

Code

```
; Run this code with update frequency 1000

startReading:
  MOV R7, #0          ; number of keys read till now
  MOV R1, #2          ; | to store sum of the two numbers
                      ; | offset of 2 is provided because
                      ; | the keys in problem statement have
                      ; | range (1, 12) instead for normal
                      ; | (0, 11)

loopRead:
  CJNE R7, #2, scanKey   ; reading two numbers
  JMP configTransmission

scanKey:
  MOV R0, #0          ; | R0 stores the key number being checked
                      ; | and will be locked when a pressed key is detected

  CLR F0;             ; reset flag to indicate scanning is initiated

  SETB P0.3           ; row3 has been checked
  CLR P0.0            ; start checking row0
  CALL colscan        ; checking columns
```

```
  JB F0, keyfound     ; key press identified

  SETB P0.0           ; row0 has been checked
  CLR P0.1            ; start checking row1
  CALL colscan        ; checking columns
  JB F0, keyfound     ; key press identified

  SETB P0.1           ; row1 has been checked
  CLR P0.2            ; start checking row2
  CALL colscan        ; checking columns
  JB F0, keyfound     ; key press identified

  SETB P0.2           ; row2 has been checked
  CLR P0.3            ; start checking row3
  CALL colscan        ; checking columns
  JB F0, keyfound      ; key press identified

  JMP loopRead

colscan:
  JB P0.4, colscan2
  SETB F0
  JNB P0.4, $
  RET

colscan2:
  INC R0
  JB P0.5, colscan3
  SETB F0
  JNB P0.5, $
  RET

colscan3:
  INC R0
  JB P0.6, colnotfound
  SETB F0
  JNB P0.6, $
  RET

colnotfound:
  INC R0
  RET



keyfound:
  MOV A, R0           ; move current index of pressed key to A
  ADD A, R1           ; add previous sum to A
  MOV R1, A           ; put current sum to R1
```

```
  INC R7              ; increment key counter
  JMP loopRead

configTransmission:
  ; trasmitting the stored keys via UART

  ; dictionary mapping for keys

  MOV 40H, #'0'       ; key value for R0 = 0x00
  MOV 41H, #'1'       ; key value for R0 = 0x01
  MOV 42H, #'2'       ; key value for R0 = 0x02
  MOV 43H, #'3'       ; key value for R0 = 0x03
  MOV 44H, #'4'       ; key value for R0 = 0x04
  MOV 45H, #'5'       ; key value for R0 = 0x05
  MOV 46H, #'6'       ; key value for R0 = 0x06
  MOV 47H, #'7'       ; key value for R0 = 0x07
  MOV 48H, #'8'       ; key value for R0 = 0x08
  MOV 49H, #'9'       ; key value for R0 = 0x09
  MOV 4AH, #'A'       ; key value for R0 = 0x0A
  MOV 4BH, #'B'       ; key value for R0 = 0x0B

  ; configuring for transmission
  CLR SM0             ; |
  SETB SM1            ; | put serial port in 8-bit UART mode

  MOV A, PCON         ; |
  SETB ACC.7          ; |
  MOV PCON, A         ; | set SMOD in PCON to double baud rate

  MOV TMOD, #20H      ; put timer 1 in 8-bit auto-reload interval timing mode
  MOV TH1, #243       ; put -13 in timer 1 high byte (timer will overflow every 13 us)
  MOV TL1, #243       ; put same value in low byte so when timer is first started it will
overflow after 13 us
  SETB TR1            ; start timer 1

initTransmission:
  MOV A, R1           ; converting the number to two BCDs and sending them to UART Reciever
  MOV B, #10
  DIV AB
  ADD A, #40H         ; |
  MOV R1, A           ; |
  MOV A, @R1          ; | decoding key value from key index
  MOV SBUF, A         ; move data to be sent to the serial port
  JNB TI, $           ; wait for TI to be set, indicating serial port has finished sending
byte
  CLR TI              ; clear TI

  MOV A, B
```

```
  ADD A, #40H
  MOV R1, A
  MOV A, @R1
  MOV SBUF, A         ; move data to be sent to the serial port
  JNB TI, $           ; wait for TI to be set, indicating serial port has finished sending
byte
  CLR TI              ; clear TI

configReceiver:
  MOV 52H, #0
  MOV 51H, #0
  MOV 50H, #0
  SETB REN                ; enable serial port receiver
  MOV R0, #50H            ; characters received from serial port will be stored in RAM
starting with address 50H

initReceiver:
  JNB RI, $               ; wait for byte to be received
  CLR RI                  ; clear the RI flag
  MOV A, SBUF             ; move received byte to A
  CJNE A, #0DH, loopReceiver ; compare it with 0DH - it it's not, skip next instruction
  JMP initLCD             ; if it is the terminating character, jump to the LCD module

loopReceiver:
  MOV @R0,A               ; move from A to location pointed to by R0
  INC R0                  ; increment R0 to point at next location where data will be
stored
  JMP initReceiver

initLCD:
  call configureFor4BitOperation
  call incrementCursorMode
  call displayOnCursonOnBlinkingOn
  call main


configureFor4BitOperation:
  ; for configuring 4-bit operation in LCD

  clr P1.3     ; instruction flow mode

  clr P1.7     ; |
  clr P1.6     ; |
  setb P1.5    ; |
  clr P1.4     ; | high nibble set

  setb P1.2    ; |
  clr P1.2     ; | negative edge
```

```
  call delay

  setb P1.2    ; |
  clr P1.2     ; | negative edge

  setb P1.7

  setb P1.2    ; |
  clr P1.2     ; | negative edge

  call delay

  ret

incrementCursorMode:
  ; for displaying next character on adjacent display
  ; Code = 0x06 = 0000 0110

  clr P1.3   ; instruction mode on

  clr P1.7   ; |
  clr P1.6   ; |
  clr P1.5   ; |
  clr P1.4   ; | high nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  setb P1.6  ; |
  setb P1.5  ; | low nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  call delay

  ret

displayOnCursonOnBlinkingOn:
  ; turning on the display and cursor and choosing blinking
  ; Code = 0x0F = 0000 1111

  clr P1.3   ; instruction mode on

  clr P1.7   ; |
  clr P1.6   ; |
  clr P1.5   ; |
```

```
  clr P1.4   ; | high nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  setb P1.7  ; |
  setb P1.6  ; |
  setb P1.5  ; |
  setb P1.4  ; | low nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  call delay

  ret


main:
  SETB P1.3     ; clear RS - indicates that data is being sent to module
  MOV R1, #50H  ; data to be sent to LCD is stored in 8051 RAM, starting at location 30H
  acall loop

loop:
  MOV A, @R1          ; move data pointed to by R1 to A
  JZ finish           ; if A is 0, then end of data has been reached - jump out of loop
  CALL sendCharacter  ; send data in A to LCD module
  INC R1              ; point to next piece of data
  JMP loop            ; repeat

finish:
  JMP $

sendCharacter:
  MOV C, ACC.7    ; |
  MOV P1.7, C     ; |
  MOV C, ACC.6    ; |
  MOV P1.6, C     ; |
  MOV C, ACC.5    ; |
  MOV P1.5, C     ; |
  MOV C, ACC.4    ; |
  MOV P1.4, C     ; | high nibble set

  SETB P1.2       ; |
  CLR P1.2        ; | negative edge on E

  MOV C, ACC.3    ; |
  MOV P1.7, C     ; |
```

```
  MOV C, ACC.2     ; |
  MOV P1.6, C      ; |
  MOV C, ACC.1     ; |
  MOV P1.5, C      ; |
  MOV C, ACC.0     ; |
  MOV P1.4, C      ; | low nibble set

  SETB P1.2        ; |
  CLR P1.2         ; | negative edge on E

  CALL delay       ; wait for BF to clear

delay:
  MOV R0, #50
  DJNZ R0, $
  RET
```

Discussion

1 LCD Module

- The LCD module consists of 16 rows and 2 columns of 5x8 dot matrices.
- Name of the pins and their corresponding functions are as follows:

Pin Name	Function
VSS	Must be grounded
VCC	5V DC power supply
RS	Register Selection
R/W	Read/write
E	Enable
DB[7:0]	Data

- From the circuit diagram, it can be observed that P1.3 is the RS of the LCD display, and P1.2 is the E of the LCD display.
- To execute any set of command, a negative edge has to be generated by E, i.e., set P1.2 to logic high and then to logic low and add some delay
- There are two types of register modes:
  - **Command mode**: Indicates flow of instruction to the LCD module, RS (P1.3) must be set to logic low to select this register mode
  - **Data mode**: Indicates flow of data to the LCD module, RS (P1.3) must be set to logic high to select this register mode
- Every operation linked with LCD display has a unique hexadecimal code. Some of them are:

Code (in hexadecimal)	Operation
0F	LCD ON, cursor ON, blinking ON
01	Clear screen
02	Return home
04	Decrement cursor
06	Increment cursor
0E	Display ON, cursor OFF
80	Force cursor to the beginning of 1st line
C0	Force cursor to the beginning of 2nd line
38	Use 2 lines and 5x7 matrix
83	Cursor line 1 position 3
3C	Activate second line
08	Display OFF, cursor OFF
C1	Jump to second line, position 1
C2	Jump to second line, position 2
0C	Display ON, cursor OFF

- To perform any operation with code, say 0x75 = 01110101B, we divide the instruction into high nibble set (0111 in this case) and low nibble set (0101 in this case). Then we do:

```
CLR P1.7     ;  0|
SETB P1.6    ;  1|
SETB P1.5    ;  1|
SETB P1.4    ;  1| high nibble set
SETB P1.2    ;   |
CLR P1.2     ;   | negative edge on E
CLR P1.7     ;  0|
SETB P1.6    ;  1|
CLR P1.5     ;  0|
SETB P1.4    ;  1| low nibble set
SETB P1.2    ;   |
```

```
CLR P1.2     ;   | negative edge on E
CALL delay
```

- The characters of my name are stored in RAM in from 0x30 to 0x37
- sendCharacter module is used for displaying the current character on the LCD display

2 Scanning Keypad

- While no key is pressed the program scans row0, row1, row2, row3 and back to row0, continuously.
- When a key is pressed the key number is placed in R0.
- For this program, the keys are numbered as:

```
+----+----+----+
| 12 | 11 | 10 |   row3
+----+----+----+
|  9 |  8 |  7 |   row2
+----+----|----+
|  6 |  5 |  4 |   row1
+----+----+----+
|  3 |  2 |  1 |   row0
+----+----+----+
 col2 col1 col0
```

- The pressed key number will be stored in R0. Therefore, R0 is initially cleared.
- Each key is scanned, and if it is not pressed R0 is incremented. In that way, when the pressed key is found, R0 will contain the key's number.
- The general-purpose flag, F0, is used by the column-scan subroutine to indicate whether or not a pressed key was found in that column. If, after returning from colScan, F0 is set, this means the key was found.
- The sum of the two numbers is stored in the register R1, which is then converted to BCD before transmitting on serial port.

3 UART Transmitter / Receiver

- The data stored in register (BCD) is transmitted via serial port. For this, the serial port is put to 8-bit UART mode. The data is transmitted at 4800 baud rate. To generate this baud rate, timer 1 must overflow every 13 us with SMOD equal to 1.
- While receiving on serial port, the data is stored on RAM starting from address 50H. 51H and 52H are null-terminated because the number of digits in the received sum is not known apriori.
- When received via serial port, the sum is displayed on LCD module.

Part 3 – Display month on LCD module via UART Tx-Rx

Objective

Transmit the corresponding month in a year on serial port and display it on LCD after receiving it on serial port. For example, for key '1' LCD output is January, key '12' LCD output is December and so on.

Circuit Diagram

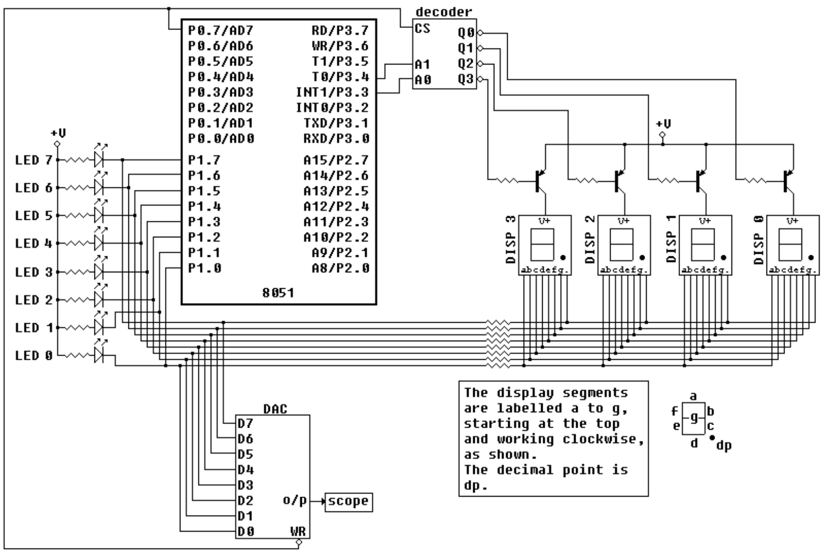

Fig. 3. Circuit diagram

Code

```
; Run this code with update frequency 1000

scanKey:
  MOV R0, #0          ; | R0 stores the key number being checked
                      ; | and will be locked when a pressed key is detected

  CLR F0;             ; reset flag to indicate scanning is initiated

  SETB P0.3           ; row3 has been checked
  CLR P0.0            ; start checking row0
  CALL colscan        ; checking columns
  JB F0, keyfound     ; key press identified

  SETB P0.0           ; row0 has been checked
  CLR P0.1            ; start checking row1
  CALL colscan        ; checking columns
  JB F0, keyfound     ; key press identified

  SETB P0.1           ; row1 has been checked
  CLR P0.2            ; start checking row2
  CALL colscan        ; checking columns
  JB F0, keyfound     ; key press identified
```

```
  SETB P0.2           ; row2 has been checked
  CLR P0.3            ; start checking row3
  CALL colscan        ; checking columns
  JB F0, keyfound      ; key press identified

  JMP scanKey

colscan:
  JB P0.4, colscan2
  SETB F0
  JNB P0.4, $
  RET

colscan2:
  INC R0
  JB P0.5, colscan3
  SETB F0
  JNB P0.5, $
  RET

colscan3:
  INC R0
  JB P0.6, colnotfound
  SETB F0
  JNB P0.6, $
  RET

colnotfound:
  INC R0
  RET


keyfound:
  MOV A,R0;
  MOV R1,A;

decodeMonth:
  CJNE R1, #0, feb
  MOV 40H, #'J'
  MOV 41H, #'A'
  MOV 42H, #'N'
  MOV 43H, #'U'
  MOV 44H, #'A'
  MOV 45H, #'R'
  MOV 46H, #'Y'
  MOV 47H, #0
  JMP configTransmission
```

```
feb:
  CJNE R1, #1, march
  MOV 40H, #'F'
  MOV 41H, #'E'
  MOV 42H, #'B'
  MOV 43H, #'R'
  MOV 44H, #'U'
  MOV 45H, #'A'
  MOV 46H, #'R'
  MOV 47H, #'Y'
  MOV 48H, #0
  JMP configTransmission

march:
  CJNE R1, #2, april
  MOV 40H, #'M'
  MOV 41H, #'A'
  MOV 42H, #'R'
  MOV 43H, #'C'
  MOV 44H, #'H'
  MOV 45H, #0
  JMP configTransmission

april:
  CJNE R1, #3, may
  MOV 40H, #'A'
  MOV 41H, #'P'
  MOV 42H, #'R'
  MOV 43H, #'I'
  MOV 44H, #'L'
  MOV 45H, #0
  JMP configTransmission

may:
  CJNE R1, #4, june
  MOV 40H, #'M'
  MOV 41H, #'A'
  MOV 42H, #'Y'
  MOV 43H, #0
  JMP configTransmission

june:
  CJNE R1, #5, july
  MOV 40H, #'J'
  MOV 41H, #'U'
  MOV 42H, #'N'
  MOV 43H, #'E'
```

```
  MOV 44H, #0
  JMP configTransmission

july:
  CJNE R1, #6, august
  MOV 40H, #'J'
  MOV 41H, #'U'
  MOV 42H, #'L'
  MOV 43H, #'Y'
  MOV 44H, #0
  JMP configTransmission

august:
  CJNE R1, #7, september
  MOV 40H, #'A'
  MOV 41H, #'U'
  MOV 42H, #'G'
  MOV 43H, #'U'
  MOV 44H, #'S'
  MOV 45H, #'T'
  MOV 46H, #0
  JMP configTransmission

september:
  CJNE R1, #8, october
  MOV 40H, #'S'
  MOV 41H, #'E'
  MOV 42H, #'P'
  MOV 43H, #'T'
  MOV 44H, #'E'
  MOV 45H, #'M'
  MOV 46H, #'B'
  MOV 47H, #'E'
  MOV 48H, #'R'
  MOV 49H, #0
  JMP configTransmission

october:
  CJNE R1, #9, november
  MOV 40H, #'O'
  MOV 41H, #'C'
  MOV 42H, #'T'
  MOV 43H, #'O'
  MOV 44H, #'B'
  MOV 45H, #'E'
  MOV 46H, #'R'
  MOV 47H, #0
  JMP configTransmission
```

```
november:
  CJNE R1, #10, december
  MOV 40H, #'N'
  MOV 41H, #'O'
  MOV 42H, #'V'
  MOV 43H, #'E'
  MOV 44H, #'M'
  MOV 45H, #'B'
  MOV 46H, #'E'
  MOV 47H, #'R'
  MOV 48H, #0
  JMP configTransmission

december:
  MOV 40H, #'D'
  MOV 41H, #'E'
  MOV 42H, #'C'
  MOV 43H, #'E'
  MOV 44H, #'M'
  MOV 45H, #'B'
  MOV 46H, #'E'
  MOV 47H, #'R'
  MOV 48H, #0
  JMP configTransmission

configTransmission:
  CLR SM0            ; |
  SETB SM1           ; | put serial port in 8-bit UART mode

  MOV A, PCON        ; |
  SETB ACC.7         ; |
  MOV PCON, A        ; | set SMOD in PCON to double baud rate

  MOV TMOD, #20H     ; put timer 1 in 8-bit auto-reload interval timing mode
  MOV TH1, #243      ; put -13 in timer 1 high byte (timer will overflow every 13 us)
  MOV TL1, #243      ; put same value in low byte so when timer is first started it will
overflow after 13 us
  SETB TR1           ; start timer 1

  MOV R0, #40H       ; number of characters transmitted till now

loopTransmission:
  MOV A, @R0
  JZ configReceiver  ; if A contains null value, stop transmission and configure receiver
  MOV SBUF, A        ; move data to be sent to the serial port
  JNB TI, $          ; wait for TI to be set, indicating serial port has finished sending
byte
```

```
  CLR TI              ; clear TI
  INC R0              ; increment R0 to point at next byte of data to be sent
  JMP loopTransmission

configReceiver:
  SETB REN                ; enable serial port receiver
  MOV 50H, #0
  MOV 51H, #0
  MOV 52H, #0
  MOV 53H, #0
  MOV 54H, #0
  MOV 55H, #0
  MOV 56H, #0
  MOV 57H, #0
  MOV 58H, #0
  MOV 59H, #0
  MOV R0, #50H            ; characters received from serial port will be stored in RAM
starting with address 50H

initReceiver:
  JNB RI, $               ; wait for byte to be received
  CLR RI                  ; clear the RI flag
  MOV A, SBUF             ; move received byte to A
  CJNE A, #0DH, loopReceiver ; compare it with 0DH - it it's not, skip next instruction
  JMP initLCD             ; if it is the terminating character, jump to the LCD module

loopReceiver:
  MOV @R0,A               ; move from A to location pointed to by R0
  INC R0                  ; increment R0 to point at next location where data will be
stored
  JMP initReceiver

initLCD:
  call configureFor4BitOperation
  call incrementCursorMode
  call displayOnCursonOnBlinkingOn
  call main


configureFor4BitOperation:
  ; for configuring 4-bit operation in LCD

  clr P1.3     ; instruction flow mode

  clr P1.7     ; |
  clr P1.6     ; |
  setb P1.5    ; |
  clr P1.4     ; | high nibble set
```

```
  setb P1.2    ; |
  clr P1.2     ; | negative edge

  call delay

  setb P1.2    ; |
  clr P1.2     ; | negative edge

  setb P1.7

  setb P1.2    ; |
  clr P1.2     ; | negative edge

  call delay

  ret

incrementCursorMode:
  ; for displaying next character on adjacent display
  ; Code = 0x06 = 0000 0110

  clr P1.3   ; instruction mode on

  clr P1.7   ; |
  clr P1.6   ; |
  clr P1.5   ; |
  clr P1.4   ; | high nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  setb P1.6  ; |
  setb P1.5  ; | low nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  call delay

  ret

displayOnCursonOnBlinkingOn:
  ; turning on the display and cursor and choosing blinking
  ; Code = 0x0F = 0000 1111

  clr P1.3   ; instruction mode on
```

```
  clr P1.7   ; |
  clr P1.6   ; |
  clr P1.5   ; |
  clr P1.4   ; | high nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  setb P1.7  ; |
  setb P1.6  ; |
  setb P1.5  ; |
  setb P1.4  ; | low nibble set

  setb P1.2  ; |
  clr P1.2   ; | negative edge

  call delay

  ret


main:
  SETB P1.3     ; clear RS - indicates that data is being sent to module
  MOV R1, #50H  ; data to be sent to LCD is stored in 8051 RAM, starting at location 30H
  acall loop

loop:
  MOV A, @R1          ; move data pointed to by R1 to A
  JZ finish           ; if A is 0, then end of data has been reached - jump out of loop
  CALL sendCharacter  ; send data in A to LCD module
  INC R1              ; point to next piece of data
  JMP loop            ; repeat

finish:
  JMP $

sendCharacter:
  MOV C, ACC.7    ; |
  MOV P1.7, C     ; |
  MOV C, ACC.6    ; |
  MOV P1.6, C     ; |
  MOV C, ACC.5    ; |
  MOV P1.5, C     ; |
  MOV C, ACC.4    ; |
  MOV P1.4, C     ; | high nibble set

  SETB P1.2       ; |
  CLR P1.2        ; | negative edge on E
```

```
  MOV C, ACC.3    ; |
  MOV P1.7, C     ; |
  MOV C, ACC.2    ; |
  MOV P1.6, C     ; |
  MOV C, ACC.1    ; |
  MOV P1.5, C     ; |
  MOV C, ACC.0    ; |
  MOV P1.4, C     ; | low nibble set

  SETB P1.2       ; |
  CLR P1.2        ; | negative edge on E

  CALL delay      ; wait for BF to clear

delay:
  MOV R0, #50
  DJNZ R0, $
  RET
```

Discussion

1 LCD Module

- The LCD module consists of 16 rows and 2 columns of 5x8 dot matrices.
- Name of the pins and their corresponding functions are as follows:

Pin Name	Function
VSS	Must be grounded
VCC	5V DC power supply
RS	Register Selection
R/W	Read/write
E	Enable
DB[7:0]	Data

- From the circuit diagram, it can be observed that P1.3 is the RS of the LCD display, and P1.2 is the E of the LCD display.
- To execute any set of command, a negative edge has to be generated by E, i.e., set P1.2 to logic high and then to logic low and add some delay
- There are two types of register modes:
    - **Command mode**: Indicates flow of instruction to the LCD module, RS (P1.3) must be set to logic low to select this register mode
    - **Data mode**: Indicates flow of data to the LCD module, RS (P1.3) must be set to logic high to select this register mode
- Every operation linked with LCD display has a unique hexadecimal code. Some of them are:

Code (in hexadecimal)	Operation
0F	LCD ON, cursor ON, blinking ON
01	Clear screen
02	Return home
04	Decrement cursor
06	Increment cursor
0E	Display ON, cursor OFF
80	Force cursor to the beginning of 1st line
C0	Force cursor to the beginning of 2nd line
38	Use 2 lines and 5x7 matrix
83	Cursor line 1 position 3
3C	Activate second line
08	Display OFF, cursor OFF
C1	Jump to second line, position 1
C2	Jump to second line, position 2
0C	Display ON, cursor OFF

- To perform any operation with code, say 0x75 = 01110101B, we divide the instruction into high nibble set (0111 in this case) and low nibble set (0101 in this case). Then we do:

```
CLR P1.7    ;  0|
SETB P1.6   ;  1|
SETB P1.5   ;  1|
SETB P1.4   ;  1| high nibble set
SETB P1.2   ;   |
CLR P1.2    ;   | negative edge on E
CLR P1.7    ;  0|
SETB P1.6   ;  1|
CLR P1.5    ;  0|
SETB P1.4   ;  1| low nibble set
SETB P1.2   ;   |
```

```
CLR P1.2     ;    | negative edge on E
CALL delay
```

- The characters of my name are stored in RAM in from 0x30 to 0x37
- sendCharacter module is used for displaying the current character on the LCD display

2 Scanning Keypad

- While no key is pressed the program scans row0, row1, row2, row3 and back to row0, continuously.
- When a key is pressed the key number is placed in R0.
- For this program, the keys are numbered as:

```
+----+----+----+
| 12 | 11 | 10 |   row3
+----+----+----+
|  9 |  8 |  7 |   row2
+----+----|----+
|  6 |  5 |  4 |   row1
+----+----+----+
|  3 |  2 |  1 |   row0
+----+----+----+
 col2 col1 col0
```

- The pressed key number will be stored in R0. Therefore, R0 is initially cleared.
- Each key is scanned, and if it is not pressed R0 is incremented. In that way, when the pressed key is found, R0 will contain the key's number.
- The general-purpose flag, F0, is used by the column-scan subroutine to indicate whether or not a pressed key was found in that column. If, after returning from colScan, F0 is set, this means the key was found.
- The key identified while scanning is then passed to decodeMonth module for initializing RAM address with characters from corresponding month string, starting with 40H.

3 UART Transmitter / Receiver

- The data stored in register (BCD) is transmitted via serial port. For this, the serial port is put to 8-bit UART mode. The data is transmitted at 4800 baud rate. To generate this baud rate, timer 1 must overflow every 13 us with SMOD equal to 1.
- While receiving on serial port, the month string is stored on RAM address starting from 50H.
- After the entire string is received, LCD module is used to display the string on display.