MPMC Programs

As per Competency-focused Outcome-based Green Curriculum-2021 (COGC-2021)

Milav Dabgar

Contents

1 MPMC Programs

1.1 Assembly Language Programs

1.1.1 MPMC GTU Paper Programs Solutions

```
ORG 0000H ; Set the starting address of the program to 0000H

MOV A, #05H    ; Load the first byte (multiplicand) into the accumulator
MOV B, #03H    ; Load the second byte (multiplier) into register B

MUL AB         ; Multiply the contents of A and B (result stored in A and B)

; At this point, the lower byte of the result is in the accumulator (A)
; and the higher byte of the result is in register B.

; Example: To store the result in memory
MOV 40H, A     ; Store the lower byte in memory location 40H
MOV 41H, B     ; Store the higher byte in memory location 41H

END            ; End of program
```

Explanation:

1. **ORG 0000H:** This directive tells the assembler that the code should be placed starting from memory address 0000H.
2. **MOV A, #05H:** This instruction loads the immediate value 05H (the first number) into the accumulator register (A).
3. **MOV B, #03H:** This instruction loads the immediate value 03H (the second number) into register B.
4. **MUL AB:** This is the core multiplication instruction. It multiplies the contents of the accumulator (A) with the contents of register B. The 16-bit result is stored across the accumulator (lower byte) and register B (higher byte).
5. **MOV 40H, A / MOV 41H, B:** These instructions are an example of how you would store the result in memory. Here, the lower byte of the result is stored at address 40H and the higher byte at 41H.

Important Notes:

- The 8051 can only multiply 8-bit numbers. The result of 05H * 03H = 0FH (15), fits within a single byte so there's no overflow in this case.
- If the result was larger than 255 (FFh), the overflow flag (OV) in the program status word (PSW) would be set.

```
ORG 0000H  ; Start the program at memory address 0000H

MOV A, 30H ; Load the first number from RAM location 30H into A
MOV B, @A  ; Move the value pointed to by A (the first number) into B

INC A      ; Increment A to point to the second number (31H)
MOV A, @A  ; Move the value pointed to by A (the second number) into A

MUL AB     ; Multiply the two numbers

MOV 52H, A ; Store the LSB of the result at memory location 52H
MOV 51H, B ; Store the MSB of the result at memory location 51H

END        ; End of program
```

Explanation:

1. **ORG 0000H:** Indicates the starting memory address for the program.
2. **MOV A, 30H:** Loads the memory address 30H into the accumulator (A).
3. **MOV B, @A:** Indirect addressing. Moves the contents of the memory location pointed to by A (which contains the first number) into register B.
4. **INC A:** Increments A to point to location 31H, where the second number is stored.
5. **MOV A, @A:** Loads the second number (from location 31H) into the accumulator.
6. **MUL AB:** Multiplies the number in the accumulator (A) with the number in register B.
7. **MOV 52H, A:** Stores the lower byte (LSB) of the result in memory location 52H.
8. **MOV 51H, B:** Stores the higher byte (MSB) of the result in memory location 51H.

```
ORG 0000H   ; Start the program at memory address 0000H

MOV A, #09H  ; Load the dividend (09h) into the accumulator
MOV B, #02H  ; Load the divisor (02h) into register B

DIV AB       ; Divide the accumulator (A) by register B
             ; Quotient will be in A, remainder in B

; Example: To store the results in memory
MOV 60H, A   ; Store the quotient in memory location 60H
MOV 61H, B   ; Store the remainder in memory location 61H

END          ; End of program
```

Explanation:

1. **ORG 0000H:** Sets the starting address of the program.
2. **MOV A, #09H:** Loads the dividend (the number to be divided) into the accumulator.
3. **MOV B, #02H:** Loads the divisor into register B.
4. **DIV AB:** Performs the division. The result (quotient) is stored in the accumulator (A), and the remainder is stored in register B.
5. **MOV 60H, A / MOV 61H, B:** These are example instructions to store the quotient and remainder in memory locations 60H and 61H, respectively.

Important Notes:

- **Integer Division:** The 8051 DIV instruction performs integer division, meaning any fractional part of the result will be discarded.
- **Overflow:** If the result of the division is too large to fit in the accumulator, the overflow flag (OV) will be set in the program status word (PSW). You'll need to add code to handle this potential overflow situation if it's relevant to your application.

Result:

In this case, 09h / 02h = 4 (quotient) with a remainder of 1.

```
ORG 0000H ; Set the program's starting address

MOV A, 20H      ; Load the address of the dividend into A
MOV B, @A       ; Move the dividend from RAM to register B
INC A           ; Increment A to point to the divisor
MOV A, @A       ; Move the divisor from RAM to the accumulator (A)

DIV AB          ; Divide the accumulator (A) by register B
                ; Quotient in A, remainder in B

MOV 40H, A      ; Store the quotient at memory location 40H
MOV 41H, B      ; Store the remainder at memory location 41H

END             ; End of program
```

Explanation:

1. **ORG 0000H:** Indicates the program starts at memory address 0000H.
2. **MOV A, 20H:** Loads the memory address 20H (where the dividend is stored) into the accumulator.
3. **MOV B, @A:** Indirect addressing. Loads the value at the memory location pointed to by A (the dividend) into register B.
4. **INC A:** Increments the accumulator to point to address 21H, where the divisor is stored.
5. **MOV A, @A:** Loads the value at the memory location pointed to by A (the divisor) into the accumulator.
6. **DIV AB:** Performs the division. The quotient is left in the accumulator (A) and the remainder in register B.
7. **MOV 40H, A:** Stores the quotient (from A) into memory location 40H.
8. **MOV 41H, B:** Stores the remainder (from B) into memory location 41H.

1.1.1.5 Write a program to create square wave of 50 % duty cycle on P1.3 pin using timer. (4) Code (Assuming Timer 0, Mode 1):

```
ORG 0000H

MOV TMOD, #01H ; Set Timer 0 in Mode 1 (16-bit timer)

; Calculate Timer Reload Value (adjust for your crystal frequency)
; Example: Assuming 12 MHz crystal frequency
; Desired period = Time for HIGH + Time for LOW = 2 * Time for HIGH
; Let's make Time for HIGH = 1 ms (adjust as needed)
; Timer count = (Crystal Frequency / 12) * Time
;             = (12000000 /12) * 0.001 = 1000
; Reload Value = 65536 - Timer Count = 65536 - 1000 = 64536
; Split into higher and lower bytes:
MOV TH0, #0xFC    ; Higher byte of reload value
MOV TL0, #0x18    ; Lower byte of reload value

SETB P1.3    ; Initially set the pin HIGH
SETB TR0     ; Start Timer 0

HERE:
JNB TF0, HERE   ; Wait for Timer 0 overflow
CLR TF0         ; Clear the overflow flag
CPL P1.3        ; Toggle the pin
SJMP HERE       ; Repeat indefinitely
```

Explanation:

1. **ORG 0000H:** Sets the program starting address.
2. **MOV TMOD, #01H:** Configures Timer 0 to operate in Mode 1 (16-bit mode).
3. **Timer Reload Value Calculation:**
   - You'll need to adjust the calculation based on your crystal frequency and desired square wave period. The example assumes a 12 MHz crystal and aims for a 1 ms HIGH time (and 1 ms LOW time).
4. **MOV TH0, #0xFC / MOV TL0, #0x18:** Load the calculated reload value into Timer 0's high and low registers.
5. **SETB P1.3:** Initially set the output pin P1.3 to HIGH.
6. **SETB TR0:** Start running Timer 0.
7. **HERE / JNB TF0, HERE:** Create a loop waiting for the Timer 0 overflow flag (TF0) to be set.
8. **CLR TF0:** Clear the Timer 0 overflow flag.
9. **CPL P1.3:** Complement (toggle) the logic level of the P1.3 pin.
10. **SJMP HERE:** Jump back to the beginning of the loop, creating an infinite square wave generation.

Important Considerations:

- **Crystal Frequency:** Make sure to adjust the reload value calculation based on your specific crystal frequency.
- **Timer Mode:** Mode 1 is a common choice for square wave generation.
- **Output Pin:** Ensure that P1.3 is configured as an output pin.
- **Desired Period/Frequency:** Adjust the calculation to achieve your specific square wave timing requirements.

```
ORG 0000H  ; Set program origin

MOV TMOD, #01H  ; Configure Timer 0 in Mode 1 (16-bit mode)

; Calculate Timer Reload Value (adjust for your desired frequency)
; Example: Assuming 11.0592 MHz crystal frequency for a 1 kHz square wave
; 1 kHz square wave has a period of 1 ms (0.001 seconds)
; Time for HIGH = Time for LOW = 0.5 ms (0.0005 seconds)
; Timer count = (Crystal Frequency / 12) * Time
;             = (11059200 / 12) * 0.0005
;             = 460.8
; Reload Value = 65536 - Timer Count = 65536 - 460 = 65076
; Split into higher and lower bytes:
MOV TH0, #0xFC    ; Higher byte of reload value (65076)
MOV TL0, #0x18    ; Lower byte of reload value (65076)

SETB P1.1    ; Initially set P1.1 HIGH
SETB TR0     ; Start Timer 0

HERE:
```

```
JNB TF0, HERE   ; Wait for Timer 0 to overflow
CLR TF0         ; Clear the overflow flag
CPL P1.1        ; Toggle the P1.1 pin
SJMP HERE       ; Loop back to create the square wave
```

Explanation:

1. **ORG 0000H:** Sets the starting address of your program.
2. **MOV TMOD, #01H:** Configures Timer 0 to operate in Mode 1, which is a 16-bit timer mode.
3. **Timer Reload Calculation:**
    - **You need to adjust this calculation based on your crystal frequency and the desired frequency of the square wave.**
    - The example aims for a 1 kHz square wave with a 11.0592 MHz crystal.
4. **MOV TH0, #0xFC / MOV TL0, #0x18:** Loads the calculated reload value into Timer 0's registers.
5. **SETB P1.1:** Sets the P1.1 pin HIGH initially.
6. **SETB TR0:** Starts Timer 0.
7. **HERE / JNB TF0, HERE:** Creates a loop that waits for the Timer 0 overflow flag (TF0).
8. **CLR TF0:** Clears the overflow flag.
9. **CPL P1.1:** Toggles the state of the P1.1 pin, changing it from HIGH to LOW or vice versa.
10. **SJMP HERE:** Jumps back to the label "HERE," creating a continuous square wave.

Key Points:

- **Crystal Frequency:** Replace the crystal frequency in the calculation with the actual value for your 8051 system.
- **Desired Frequency:** Modify the calculation to get the square wave frequency you need.
- **Pin Configuration:** Ensure that P1.1 is configured as an output pin.

```
ORG 0000H

DELAY:
MOV R0, #50D    ; Load a value into register R0 (adjust for delay)
MOV R1, #250D   ; Load a value into register R1 (adjust for delay)

DJNZ_R1:        ; Decrement R1 and jump if not zero
DJNZ R1, DJNZ_R1

DJNZ_R0:        ; Decrement R0 and jump if not zero
DJNZ R0, DJNZ_R0

RET             ; Return from the subroutine
```

Explanation:

1. **DELAY:** This label marks the beginning of your delay subroutine.
2. **MOV R0, #50D / MOV R1, #250D:** These instructions load initial values into two registers, R0 and R1. The larger the values, the longer the delay. You'll need to adjust these based on your crystal frequency and the desired delay length.
3. **DJNZ_R1 / DJNZ_R0:** These are "Decrement and Jump if Not Zero" instructions. They form two nested loops:
    - **Outer Loop (R0):** The outer loop decrements R0 and repeats until R0 reaches zero.

- **Inner Loop (R1):** The inner loop decrements R1 and repeats until R1 reaches zero. Each time the inner loop runs, it reloads R1 with its initial value.

How it Works:

The nested loops create a series of decrement operations. The combination of instructions and the initial values in R0 and R1 determine the overall time the delay takes to execute.

Important Considerations:

- **Accuracy:** Software delays are not perfectly precise. Their timing depends on your crystal frequency and the number of instructions within the loop.
- **Crystal Frequency:** For more accurate delays, you'll need to calibrate the initial values (in R0 and R1) based on your crystal frequency.
- **Timer Alternatives:** For very precise delays, consider using the 8051's built-in timers instead of software delay loops.

```
MOV A, R2        ; Copy the contents of R2 into the accumulator
ANL A, #0F0H     ; Perform a logical AND with 0F0H to mask the lower 4 bits
MOV R2, A        ; Move the result back into R2
```

Explanation:

1. **MOV A, R2:** This instruction copies the current value stored in register R2 into the accumulator (A).
2. **ANL A, #0F0H:** This performs a logical AND operation between the value in the accumulator (which now contains the original value of R2) and the hexadecimal value 0F0H. The result will be that:
   - Bits 4-7 of the accumulator will remain unchanged.
   - Bits 0-3 of the accumulator will become 0 (masked).
3. **MOV R2, A:** This instruction moves the modified contents of the accumulator (with the lower bits masked) back into register R2.

Key Points

- **Hexadecimal Mask:** The value 0F0H (binary: 1111 0000) is used as a mask because it has '1's in the bit positions you want to preserve and '0's in the bit positions you want to clear.
- **Masking:** Masking is a technique used to isolate or clear specific bits within a byte.

```
MOV R0, #20H   ; Load starting address into a register (R0 in this case)
MOV A, #0FFH   ; Load the data to be filled into the accumulator

FILL_LOOP:
MOV @R0, A     ; Store the content of the accumulator into the memory
    ↪ location pointed to by R0
INC R0         ; Increment R0 to point to the next memory location
CJNE R0, #30H, FILL_LOOP  ; Compare R0 with the ending address + 1 (30H),
    ↪ jump to FILL_LOOP if not equal
```

Explanation:

1. **MOV R0, #20H:** Load the starting memory address (20H) into a register (we're using R0).

2. **MOV A, #0FFH:** Load the value you want to fill (FFH) into the accumulator.
3. **FILL_LOOP:** This label marks the beginning of the loop.
4. **MOV @R0, A:** This instruction uses indirect addressing. It stores the contents of the accumulator (FFH) into the memory location pointed to by the register R0.
5. **INC R0:** Increment the register R0 to point to the next memory location.
6. **CJNE R0, #30H, FILL_LOOP:** This instruction means “Compare and Jump if Not Equal.” It compares the contents of R0 with the value 30H (which is the ending address + 1). If they are not equal, the program jumps back to the FILL_LOOP label, continuing the filling process.

Key Points:

- **Indirect Addressing:** The @R0 syntax means that the contents of R0 are used as the memory address.
- **Loop Termination:** The CJNE instruction ensures the loop runs until memory location 2FH is filled.

```
ORG 0000H  ; Set origin of the program

; Load data from external memory
MOV A, #30H      ; Load lower byte of the first number's address into A
MOVC A, @A+DPTR  ; Fetch the first number from external memory using DPTR
MOV B, A          ; Store the first number in register B

MOV A, #31H      ; Load lower byte of the second number's address into A
MOVC A, @A+DPTR  ; Fetch the second number from external memory

ADD A, B          ; Add the two numbers (result now in A)

MOV A, #32H      ;  Load lower byte of the result address into A
MOVX @DPTR, A    ; Store the result in external memory using DPTR

END              ; End of program
```

Explanation:

1. **ORG 0000H:** Sets the starting memory address for the program.
2. **MOV A, #30H / MOVC A, @A+DPTR:**
   - Loads the lower byte of the first number’s address (2030H) into the accumulator (A).
   - Uses the DPTR (Data Pointer) register to access external memory. The MOVC instruction fetches the byte at the address calculated by adding the contents of A to the value in DPTR.
3. **MOV B, A:** Stores the fetched first number in register B.
4. **MOV A, #31H / MOVC A, @A+DPTR** Repeats the process to fetch the second number from address 2031H.
5. **ADD A, B:** Adds the two numbers together, storing the result in the accumulator (A).
6. **MOV A, #32H / MOVX @DPTR, A:**
   - Loads the lower byte of the result’s address (2032H) into A.
   - Uses MOVX for external memory access, storing the result from A at the address pointed to by DPTR.

Important Notes:

- **DPTR Setup:** Ensure that your DPTR register is correctly initialized to point to the start of external memory before executing this code.
- **MOVC vs. MOVX:** MOVC is used to read from code memory (usually within the 8051), while MOVX is used for external data memory.

1.1.1.11 Draw circuit diagram for interfacing 8 LEDS on port 1. Write a program to flash LEDS in sequence (on 1 LED at a time) with suitable time delay. (7) Circuit Diagram

1. **8051 Microcontroller:** The heart of the circuit. Choose your specific 8051 microcontroller model.
2. **LEDs:** 8 regular LEDs (choose a suitable color).
3. **Current-Limiting Resistors:** One resistor for each LED. Calculate the resistor value using this formula:

   ```
   Resistor Value (Ohms) = (Supply Voltage - LED Forward Voltage) /
       ↪ Desired LED Current
   ```

   - Typical forward voltage for LEDs is around 1.8V - 3.3V (check your LED datasheet)
   - Common LED current is around 20mA (0.02A)
4. **Connections:**
   - Connect one leg of each LED to a separate pin on Port 1 of the 8051 (P1.0 - P1.7).
   - Connect the other leg of each LED to a current-limiting resistor, and then connect those resistors to ground.

Example Circuit (Schematic would be ideal, but I'll provide a textual description):

- **Assume 5V supply and standard red LEDs (2V forward voltage)**
- P1.0 —[330 Ohm Resistor]— LED — GND
- P1.1 —[330 Ohm Resistor]— LED — GND
- ... (Repeat connections the same way for P1.2 to P1.7)

8051 Program

```
ORG 0000H

START:
MOV R0, #00H   ; Initialize a counter
MOV A, #01H    ; Initial LED pattern (0000 0001)

LOOP:
MOV P1, A      ; Output the pattern to Port 1
CALL DELAY     ; Call a delay subroutine
ROR A          ; Rotate the pattern one bit to the right
INC R0         ; Increment counter
CJNE R0, #08, LOOP ; Repeat until 8 LEDs have been lit

SJMP START     ; Restart the sequence

; Simple Delay Subroutine
DELAY:
MOV R1, #200D   ; Adjust these values for
MOV R2, #00D    ; desired delay time
DJNZ R2, $
DJNZ R1, $
RET
```

Explanation

- **START:** Sets up a counter and initial LED pattern

- **LOOP:** Outputs pattern to the LEDs, calls delay, rotates the '1' bit for the next LED.
- **CJNE:** Checks if 8 shifts have occurred, restarts if not.
- **DELAY:** A basic software delay using nested loops.

Key Points

- **Port Output:** Ensure Port 1 is configured as output.
- **Resistor Calculation:** Calculate the correct resistor value for your LEDs and supply voltage.
- **Delay Adjustment:** Modify values in the DELAY subroutine for your desired LED flashing speed.

1.1.1.12 Write a program to separate data 71h stored in accumulator , in two registers R3=07h and R4=01h. (4)

We'll use a combination of bit-shifting and masking operations:

```
MOV A, #71H    ; Load 71h into the accumulator

; Extract lower 4 bits (R3)
MOV R3, A      ; Store the value of A in R3
AND A, #0FH    ; Mask off the upper 4 bits (keep only the lower nibble)

; Extract upper 4 bits (R4)
MOV R4, A      ; The accumulator now holds only the upper nibble
SHR A          ; Shift right by 4 positions (move upper nibble to lower)
```

Explanation:

1. **MOV A, #71H:** Load the value 71h into the accumulator (A).
2. **Extract lower 4 bits:**
   - **MOV R3, A:** Store the original value from the accumulator into R3. Now both the accumulator and R3 have the value 71h.
   - **AND A, #0FH:** Perform a logical AND operation with 0Fh (00001111 in binary) to mask off the upper 4 bits in the accumulator. Now, the accumulator only holds 00000111 (which is 7).
3. **Extract upper 4 bits**
   - **MOV R4, A:** Store the masked value (the upper nibble, now in the lower 4 bits) into R4.
   - **SHR A:** Shift the accumulator right by 4 positions. This moves the upper nibble (01) into the lower 4 bits, and the accumulator now holds 00000001 (which is 1).

At the end of this code:

- **R3:** Contains 00000111 (7)
- **R4:** Contains 00000001 (1)

```
ORG 0000H          ; Set the program's starting address

MOV R0, #00H        ; Initialize a register (R0) as the accumulator
MOV R1, #09H        ; Initialize a counter (R1) to keep track of numbers

LOOP:
ADD A, R1           ; Add the current number (from R1) to the accumulator
DJNZ R1, LOOP       ; Decrement the counter and jump to LOOP if not zero
MOV 77H, A          ; Store the result (sum) in memory location 77H

END                 ; End of program
```

Explanation:

1. **ORG 0000H:** Sets the starting address of your code.
2. **MOV R0, #00H:** Initializes register R0 to 0. R0 will store the running sum.
3. **MOV R1, #09H:** Initializes register R1 to 9, which will be our counter.
4. **LOOP:** This label marks the beginning of the loop.
5. **ADD A, R1:** Adds the value in R1 to the accumulator (where the running sum is stored).

6. **DJNZ R1, LOOP:** Decrements R1 and jumps back to the LOOP label if R1 is not zero. This loop continues for 9 iterations.
7. **MOV 77H, A:** After the loop, the accumulator (A) holds the sum of the first 9 numbers. This instruction stores that sum in memory location 77H.
8. **END:** Indicates the end of the program.

Key Points

- **Registers:** We use registers for calculations and as a loop counter.
- **DJNZ Instruction:** The 'Decrement and Jump if Not Zero' instruction creates the loop.

```
PUSH R1   ; Push the contents of R1 onto the stack
PUSH R2   ; Push the contents of R2 onto the stack
POP R1    ; Pop the top value from the stack (originally from R2) into R1
POP R2    ; Pop the next value from the stack (originally from R1) into R2
```

Explanation:

1. **PUSH R1:** Pushes the contents of register R1 onto the system stack.
2. **PUSH R2:** Pushes the contents of register R2 onto the stack (on top of R1's value).
3. **POP R1:** Pops the top value from the stack and stores it into R1. Since we pushed R2 last, this will be R2's original value.
4. **POP R2:** Pops the next value from the stack and stores it into R2. This will be R1's original value.

Result:

After executing this code, the values in R1 and R2 will have been effectively swapped.

Important Note:

The stack in the 8051 microcontroller operates in a LIFO (Last In, First Out) manner. This means the last value pushed onto the stack will be the first value popped off.

```
ORG 0000H   ; Set the program's starting address

MOV R0, #30H    ; Load the starting memory address into R0
MOV R1, #21     ; Initialize counter (21 locations from 30H to 50H inclusive
    ↪ )
MOV A, #99H     ; Load the data to be copied into the accumulator

COPY_LOOP:
MOV @R0, A      ; Store the data from the accumulator into the memory
    ↪ location pointed to by R0
INC R0          ; Increment R0 to point to the next memory location
DJNZ R1, COPY_LOOP ; Decrement the counter and jump back to COPY_LOOP if
    ↪ not zero

END             ; End of program
```

Explanation:

1. **ORG 0000H:** Sets the program's starting memory address in the code space.
2. **MOV R0, #30H:** Loads the starting RAM address (30H) into register R0.
3. **MOV R1, #21:** Loads the counter value into register R1. Since there are 21 memory locations from 30H to 50H (inclusive), we initialize our counter with 21.
4. **MOV A, #99H:** Loads the data (99H) to be copied into the accumulator.

5. **COPY_LOOP:** This label marks the beginning of the loop.
6. **MOV @R0, A:** Uses indirect addressing to store the contents of the accumulator (99H) into the memory location currently pointed to by R0.
7. **INC R0:** Increments R0 to point to the next memory location where the data will be copied.
8. **DJNZ R1, COPY_LOOP:** Decrements the counter in R1 and jumps back to the COPY_LOOP label if the counter is not zero. The loop continues until the counter reaches zero.

1.1.1.16 Draw a diagram to connect 8 switches with port P1 and 8 LEDs with port P2 and write a program to show status of switch on LED. (If switch is ON then LED is ON and if switch is OFF, LED is OFF). (7) Circuit Diagram

Components:

- 8051 Microcontroller
- 8 Switches (simple push-button or toggle switches)
- 8 LEDs
- 8 Current-limiting resistors (calculate the value based on your specific LEDs)
- Breadboard and connecting wires

Connections:

1. **Port P1 (Input):**
   - Connect one end of each switch to a separate pin on Port P1 (P1.0 - P1.7).
   - Connect the other end of each switch to the microcontroller's ground (GND).
2. **Port P2 (Output):**
   - Connect the anode (longer leg) of each LED to a separate pin on Port P2 (P2.0 - P2.7).
   - Connect the cathode (shorter leg) of each LED to a current-limiting resistor. Connect the other end of each resistor to ground (GND).

Important:

- **Pull-up Resistors:** You'll likely need pull-up resistors (around 10k Ohms) connected between each input pin on Port P1 and the supply voltage (VCC). This ensures a defined logic level when the switches are open.

8051 Program

```
ORG 0000H

LOOP:
MOV A, P1        ; Read the input from Port P1
MOV P2, A        ; Transfer the input directly to Port P2
SJMP LOOP        ; Jump back to continuously monitor the switches

END              ; End of program
```

Explanation

- **ORG 0000H:** Sets the program's starting address.
- **LOOP:** Label for the main program loop.
- **MOV A, P1:** Reads the entire byte from Port P1 (the status of all 8 switches) and stores it in the accumulator (A).
- **MOV P2, A:** Directly transfers the value from the accumulator to Port P2, controlling the LEDs to mirror the switch states.
- **SJMP LOOP:** Short jump back to the beginning of the loop for continuous monitoring.

Key Points

- **Switch Logic:** Make sure your switch connections result in a logic HIGH when pressed and a logic LOW when released.
- **LED Considerations:** Ensure Port P2 can handle the current requirements of your LEDs.

```
ORG 0000H   ; Set the program's starting address

MOV DPTR, #2000H  ; Initialize DPTR to point to the start of external RAM
MOV R0, #10       ; Initialize a counter to track 10 numbers
MOV A, @DPTR      ; Load the first number into the accumulator
MOV 20H, A        ; Initialize internal RAM location 20H with the first
    ↪ number (assume it's the largest initially)

LOOP:
INC DPTR          ; Move to the next number in external RAM
MOVC A, @A+DPTR   ; Fetch the current number
CJNE A, 20H, NEXT ; Compare the current number with the largest so far
MOV 20H, A        ; If the current number is larger, update the largest

NEXT:
DJNZ R0, LOOP     ; Decrement the counter and loop if not zero

END               ; End of program
```

Explanation:

1. **ORG 0000H:** Sets the starting address of the program.
2. **MOV DPTR, #2000H:** Initializes the data pointer (DPTR) to point to the start of the numbers in external RAM (2000h).
3. **MOV R0, #10:** Initializes a counter (R0) to keep track of the 10 numbers.
4. **MOV A, @DPTR / MOV 20H, A:** Loads the first number into the accumulator and also stores it in internal RAM location 20H as our initial assumption for the largest number.
5. **LOOP:** Labels the beginning of the loop.
6. **INC DPTR:** Increments DPTR to point to the next number.
7. **MOVC A, @A+DPTR:** Fetches the current number from external RAM using DPTR.
8. **CJNE A, 20H, NEXT:** Compares the current number (in A) with the assumed largest number (at memory location 20H). If they are not equal, it jumps to the NEXT label.
9. **MOV 20H, A:** If the current number is larger, replaces the content of memory location 20H (our largest number) with it.
10. **NEXT:** Label for continuing to the next number.
11. **DJNZ R0, LOOP:** Decrements the counter (R0) and jumps back to LOOP if the counter is not zero.

At the end of this program, the largest number will be stored in internal RAM location 20H.

```
ORG 0000H ; Set the starting address of the program

; Add the numbers in R0 and R1
ADD A, R0   ; Add the contents of R0 to the accumulator
MOV R1, A   ; Store the result in R1 (in case of overflow)

; Store the result in external RAM
MOV DPTR, #1030H    ; Load DPTR with the starting external RAM address
MOV A, R1           ; Move the lower byte of the result into A
MOVX @DPTR, A       ; Store the lower byte at 1030h
INC DPTR            ; Increment DPTR to point to 1031h
MOV A, R2           ; Move the higher byte of the result (if any) into A
MOVX @DPTR, A       ; Store the higher byte at 1031h

END                 ; End of program
```

Explanation:

1. **ORG 0000H:** Sets the starting address of the program.
2. **ADD A, R1:** Adds the contents of registers R0 and R1, storing the result in the accumulator (A).
3. **MOV R1, A:** Stores the result in R1 as well. This handles the case where the addition results in a carry (overflow), ensuring the MSB is stored correctly.
4. **MOV DPTR, #1030H:** Initializes the DPTR (Data Pointer) with the starting address (1030h) in external RAM.
5. **MOV A, R1 / MOVX @DPTR, A:** Moves the lower byte of the result to the accumulator and then stores it at the location pointed to by DPTR (1030h) using the MOVX instruction (for external memory access).
6. **INC DPTR / MOV A, R2 / MOVX @DPTR, A:** Increments DPTR to address 1031h, moves the higher byte (if any) of the result into the accumulator, and stores it using MOVX.

Key Points:

- **DPTR Setup:** Make sure your DPTR is correctly set up to point to the external memory region you want to use.
- **Overflow Handling:** This code correctly handles the potential overflow when adding 8-bit numbers.

1.1.1.19 Write an ALP to exchange the content of A and B (3) Method 1: Using a Temporary Register (e.g., R0)

```
MOV R0, A    ; Store the contents of A in a temporary register (R0)
MOV A, B     ; Move the contents of B into A
MOV B, R0    ; Move the original contents of A (from R0) into B
```

Method 2: Using the XCH Instruction

```
XCH A, B     ; Directly exchange the contents of A and B
```

Method 3: Using XOR Operations

```
XOR A, B     ; XOR the contents of A and B, result in A
XCH A, B     ; Exchange A and B
XOR A, B     ; XOR A and B again (result in original value of A, now in B)
```

Explanation:

- **Method 1:** This is the most general approach, using a temporary register to hold one of the values during the swap.
- **Method 2:** The XCH instruction is specifically designed for exchanging values between the accumulator and another register. It's the most efficient way if your 8051 microcontroller supports it.
- **Method 3:** This method uses the XOR (Exclusive OR) operation, which has the interesting property that when you XOR a value with itself, the result is zero. This allows for a clever exchange mechanism.

```
ORG 0000H   ; Set program origin

MUL AB      ; Multiply the accumulator (A) by register B. Result stored in A
    ↪ (LSB) and B (MSB)

END         ; End of program
```

Explanation:

- **ORG 0000H:** Indicates the starting address of the program's code.
- **MUL AB:** This is the core multiplication instruction. It multiplies the contents of the accumulator and register B. The 16-bit result is stored across both the accumulator (lower 8 bits) and register B (higher 8 bits).

Important Notes:

- **8-bit Limitation:** The 8051 can only directly multiply 8-bit numbers. If you need to multiply larger numbers, you'll have to implement a multi-byte multiplication algorithm using a series of additions and shifts.
- **Result Location:** Remember that the lower byte of the result will be in the accumulator (A) and the higher byte will be in register B after the multiplication.

Example:

If A = 5 (00000101) and B = 3 (00000011), then after MUL AB:

- A (Accumulator) would contain 15 (00001111) - the lower byte
- B would contain 0 (00000000) - the higher byte (in this case, it's zero)

```
ORG 0000H   ; Set program origin

DIV AB      ; Divide accumulator (A) by register B. Quotient in A, remainder
    ↪ in B

END         ; End of program
```

Explanation:

- **ORG 0000H:** Sets the program's starting address.
- **DIV AB:** This is the core division instruction. It divides the contents of the accumulator (which should contain the dividend) by the contents of register B (the divisor). After the division:
  - **Quotient:** Stored in the accumulator (A)
  - **Remainder:** Stored in register B

Important Notes

- **Integer Division:** The 8051's DIV instruction performs integer division, meaning any fractional part of the result will be discarded.
- **Zero Division:** Ensure that the value in register B is not zero before performing the division. Dividing by zero will cause an overflow flag (OV) to be set in the program status word (PSW).

Example:

If A = 10 (00001010) and B = 3 (00000011), then after DIV AB:

- A (Accumulator) would contain 3 (00000011) – the quotient
- B would contain 1 (00000001) – the remainder

1.1.1.22 Write a program to copy block of 8 data starting from location 100h to 200h.

Here's an assembly program for the 8051 microcontroller to copy a block of 8 bytes of data from starting location 100H to destination location 200H:

```
ORG 0000H   ; Program starts at memory location 0000H

; Initialization
MOV DPTR, #100H     ; Set DPTR to point to the source block (100H)
MOV R0, #200H       ; Set R0 to point to the destination block (200H)
MOV R1, #08H        ; Set R1 as the loop counter (8 bytes to copy)
```

```
COPY_LOOP:
MOVX A, @DPTR    ; Read a byte from the source using DPTR
MOVX @R0, A      ; Write the byte to the destination using R0
INC DPTR         ; Increment DPTR to point to the next source byte
INC R0           ; Increment R0 to point to the next destination byte
DJNZ R1, COPY_LOOP ; Decrement R1 and jump if not zero

; End of Program (You can add more code here or an infinite loop)
END
```

Explanation

1. **ORG 0000H:** This directive tells the assembler to place the code starting from memory location 0000H.
2. **Initialization:**
   - We load the Data Pointer (DPTR) with the starting address of the source block (100H).
   - Register R0 is loaded with the starting address of the destination block (200H).
   - Register R1 is initialized to 8, which is the number of bytes we want to copy.
3. **COPY_LOOP:**
   - MOVX A, @DPTR: Reads a byte from external RAM pointed to by DPTR and stores it in the accumulator.
   - MOVX @R0, A: Writes the byte from the accumulator to external RAM pointed to by R0.
   - INC DPTR, INC R0: Increment both DPTR and R0 to move to the next memory locations.
   - DJNZ R1, COPY_LOOP: Decrement R1 and jump back to the 'COPY_LOOP' label if R1 is not zero (meaning we haven't copied all 8 bytes yet).
4. **END:** Signifies the end of the assembly program.

Key Points

- This assumes you have external RAM where you are storing the data.
- You may need to adapt the addresses (100H and 200H) if your data is stored elsewhere.

1.1.1.23 Write a program to add two bytes of data and store result in R0 register.

Here's the 8051 assembly code to add two bytes of data and store the result in register R0:

```
; Data initialization - you might load these from memory in a real program
MOV A, #56H     ; Load the first byte of data into the accumulator
MOV B, #23H     ; Load the second byte of data into register B

; Addition
ADD A, B         ; Add the value in register B to the accumulator
MOV R0, A       ; Store the result (which is now in the accumulator) into R0

; End of program (you might do something with the result or add an infinite
    ↪  loop here)
END
```

Explanation

1. **Data Initialization:**
   - MOV A, #56H: Loads the immediate value 56H (hexadecimal) into the accumulator (A register).
   - MOV B, #23H: Loads the immediate value 23H into register B.
2. **Addition:**
   - ADD A, B: Adds the value in register B to the value in the accumulator. The result remains in the accumulator.
3. **Storing the Result:**

- MOV R0, A: Moves the value from the accumulator (which holds the sum) into register R0.

Important Points

- You can replace the MOV instructions with ways to get data from other sources (memory, user input, etc.).
- Make sure that the sum of your two data bytes can fit into 8 bits to avoid overflow.

1.1.2 Mazidi Book Assembly Language Programs

```
MOV A,#0 ;A=0, clear ACC
MOV R2,#10 ;load counter R2=10
AGAIN: ADD A,#03 ;add 03 to ACC
DJNZ R2,AGAIN ;repeat until R2=0,10 times
MOV R5,A ;save A in R5
```

```
MOV A,#55H ;A=55H
MOV R3,#10 ;R3=10, outer loop count
NEXT: MOV R2,#70 ;R2=70, inner loop count
AGAIN: CPL A ;complement A register
DJNZ R2,AGAIN ;repeat it 70 times
DJNZ R3,NEXT
```

```
MOV A,#0 ;A=0
MOV R5,A ;clear R5
ADD A,#79H ;A=0+79H=79H
JNC N_1 ;if CY=0, add next number
INC R5 ;if CY=1, increment R5
N_1: ADD A,#0F5H ;A=79+F5=6E and CY=1
JNC N_2 ;jump if CY=0
INC R5 ;if CY=1,increment R5 (R5=1)
N_2: ADD A,#0E2H ;A=6E+E2=50 and CY=1
JNC OVER ;jump if CY=0
INC R5 ;if CY=1, increment 5
OVER: MOV R0,A ;now R0=50H, and R5=02
```

```
ORG 0
BACK: MOV A,#55H ;load A with 55H
MOV P1,A ;send 55H to port 1
LCALL DELAY ;time delay
MOV A,#0AAH ;load A with AA (in hex)
MOV P1,A ;send AAH to port 1
LCALL DELAY
SJMP BACK ;keep doing this indefinitely

;———————— this is delay subroutine ————————
ORG 300H ;put DELAY at address 300H
DELAY: MOV R5,#0FFH ;R5=255 (FF in hex), counter
AGAIN: DJNZ R5,AGAIN ;stay here until R5 become 0
RET ;return to caller (when R5 =0)
END
```

```
ORG 0
BACK: MOV A,#55H ;load A with 55H
MOV P1,A ;send 55H to p1
MOV R4,#99H
MOV R5,#67H
LCALL DELAY ;time delay
MOV A,#0AAH ;load A with AA
MOV P1,A ;send AAH to p1
LCALL DELAY
SJMP BACK ;keeping doing this

;——————this is the delay subroutine—————
ORG 300H
DELAY: PUSH 4 ;push R4
PUSH 5 ;push R5
MOV R4,#0FFH;R4=FFH
NEXT: MOV R5,#0FFH;R5=FFH
AGAIN: DJNZ R5,AGAIN
DJNZ R4,NEXT
POP 5 ;POP into R5
POP 4 ;POP into R4
RET ;return to caller
END
```

```
BACK: MOV A,#55H
MOV P0,A
ACALL DELAY
MOV A,#0AAH
MOV P0,A
ACALL DELAY
SJMP BACK

;——————this is the delay subroutine—————
DELAY: MOV R5,#0FFH ;R5=255 (FF in hex), counter
AGAIN: DJNZ R5,AGAIN ;stay here until R5 become 0
RET ;return to caller (when R5 =0)
```

```
MOV A,#0FFH ;A=FF hex
MOV P0,A ;make P0 an i/p port
;by writing it all 1s
BACK: MOV A,P0 ;get data from P0
MOV P1,A ;send it to port 1
SJMP BACK ;keep doing it
```

```
MOV A,#55H
BACK: MOV P1,A
ACALL DELAY
CPL A
SJMP BACK

;——————this is the delay subroutine—————
DELAY: MOV R5,#0FFH ;R5=255 (FF in hex), counter
AGAIN: DJNZ R5,AGAIN ;stay here until R5 become 0
RET ;return to caller (when R5 =0)
```

```
MOV A,#0FFH ;A=FF hex
MOV P1,A ;make P1 an input port
;by writing it all 1s
MOV A,P1 ;get data from P1
MOV R7,A ;save it to in reg R7
ACALL DELAY ;wait
MOV A,P1 ;another data from P1
MOV R5,A ;save it to in reg R5

;————this is the delay subroutine———
DELAY: MOV R5,#0FFH ;R5=255 (FF in hex), counter
AGAIN: DJNZ R5,AGAIN ;stay here until R5 become 0
RET ;return to caller (when R5 =0)
```

```
HERE: SETB P1.0 ;set to high bit 0 of port 1
LCALL DELAY ;call the delay subroutine
CLR P1.0 ;P1.0=0
LCALL DELAY
SJMP HERE ;keep doing it

;;Another way to write the above program is:
;HERE: CPL P1.0 ;set to high bit 0 of port 1
;LCALL DELAY ;call the delay subroutine
;SJMP HERE ;keep doing it

;————this is the delay subroutine———
DELAY: MOV R5,#0FFH ;R5=255 (FF in hex), counter
AGAIN: DJNZ R5,AGAIN ;stay here until R5 become 0
RET ;return to caller (when R5 =0)
```

```
SETB P1.2 ;make P1.2 an input
MOV A,#45H ;A=45H
AGAIN: JNB P1.2,AGAIN ; get out when P1.2=1
MOV P0,A ;issue A to P0
SETB P2.3 ;make P2.3 high
CLR P2.3 ;make P2.3 low for H–to–L
```

```
HERE: JNB P2.3,HERE ;keep monitoring for high
SETB P1.5 ;set bit P1.5=1
CLR P1.5 ;make high–to–low
SJMP HERE ;keep repeating
```

```
SETB P1.7 ;make P1.7 an input
AGAIN: JB P1.2,OVER ;jump if P1.7=1
MOV P2,'N' ;SW=0, issue 'N' to P2
SJMP AGAIN ;keep monitoring
OVER: MOV P2,#'Y' ;SW=1, issue 'Y' to P2
SJMP AGAIN ;keep monitoring
```

```
SETB P1.7 ;make P1.7 an input
AGAIN: MOV C,P1.2 ;read SW status into CF
JC OVER ;jump if SW=1
MOV P2,#'N' ;SW=0, issue 'N' to P2
SJMP AGAIN ;keep monitoring
OVER: MOV P2,#'Y' ;SW=1, issue 'Y' to P2
SJMP AGAIN ;keep monitoring
```

```
SETB P1.7 ;make P1.7 an input
AGAIN: MOV C,P1.0 ;read SW status into CF
MOV P2.7,C ;send SW status to LED
SJMP AGAIN ;keep repeating
```

1.1.2.16 Example 5-1 Write code to send 55H to ports P1 and P2, using (a) their names (b) their addresses Solution :

```
;(a)
MOV A,#55H ;A=55H
MOV P1,A ;P1=55H
MOV P2,A ;P2=55H
```

```
;(b) From Table 5-1, P1 address=80H; P2 address=A0H
MOV A,#55H ;A=55H
MOV 80H,A ;P1=55H
MOV 0A0H,A ;P2=55H
```

1.1.2.17 Example 5-2 Show the code to push R5 and A onto the stack and then pop them back them into R2 and B, where B = A and R2 = R5 Solution:

```
PUSH 05 ;push R5 onto stack
PUSH 0E0H ;push register A onto stack
POP 0F0H ;pop top of stack into B
;now register B = register A
POP 02 ;pop top of stack into R2
;now R2=R6
```

1.1.2.18 Example 5-3 Write a program to copy the value 55H into RAM memory locations 40H to 41H using (a) direct addressing mode, (b) register indirect addressing mode without a loop, and (c) with a loop Solution:

```
;(a)
MOV A,#55H ;load A with value 55H
MOV 40H,A ;copy A to RAM location 40H
MOV 41H,A ;copy A to RAM location 41H
;(b)
MOV A,#55H ;load A with value 55H
MOV R0,#40H ;load the pointer. R0=40H
MOV @R0,A ;copy A to RAM R0 points to
INC R0 ;increment pointer. Now R0=41h
MOV @R0,A ;copy A to RAM R0 points to
;(c)
MOV A,#55H ;A=55H
MOV R0,#40H ;load pointer.R0=40H,
MOV R2,#02 ;load counter, R2=3
AGAIN: MOV @R0,A ;copy 55 to RAM R0 points to
INC R0 ;increment R0 pointer
DJNZ R2,AGAIN ;loop until counter = zero
```

1.1.2.19 Example 5-4 Write a program to clear 16 RAM locations starting at RAM address 60H Solution:

```
CLR A ;A=0
MOV R1,#60H ;load pointer. R1=60H
MOV R7,#16 ;load counter, R7=16
AGAIN: MOV @R1,A ;clear RAM R1 points to
INC R1 ;increment R1 pointer
DJNZ R7,AGAIN ;loop until counter=zero
```

1.1.2.20 Example 5-5 Write a program to copy a block of 10 bytes of data from 35H to 60H Solution:

```
MOV R0,#35H ;source pointer
MOV R1,#60H ;destination pointer
MOV R3,#10 ;counter
BACK: MOV A,@R0 ;get a byte from source
MOV @R1,A ;copy it to destination
INC R0 ;increment source pointer
INC R1 ;increment destination pointer
DJNZ R3,BACK ;keep doing for ten bytes
```

1.1.2.21 Example 5-6 In this program, assume that the word "USA" is burned into ROM locations starting at 200H. And that the program is burned into ROM locations starting at 0. Analyze how the program works and state where "USA" is stored after this program is run. Solution:

```
ORG 0000H ;burn into ROM starting at 0
MOV DPTR,#200H ;DPTR=200H look-up table addr
CLR A ;clear A(A=0)
MOVC A,@A+DPTR ;get the char from code space
MOV R0,A ;save it in R0
INC DPTR ;DPTR=201 point to next char
CLR A ;clear A(A=0)
MOVC A,@A+DPTR ;get the next char
MOV R1,A ;save it in R1
INC DPTR ;DPTR=202 point to next char
CLR A ;clear A(A=0)
MOVC A,@A+DPTR ;get the next char
MOV R2,A ;save it in R2
Here: SJMP HERE ;stay here

;Data is burned into code space starting at 200H
ORG 200H
MYDATA:DB "USA"
END ;end of program
```

1.1.2.22 Example 5-8 Write a program to get the x value from P1 and send x2 to P2, continuously Solution:

```
ORG 0
MOV DPTR,#300H ;LOAD TABLE ADDRESS
MOV A,#0FFH ;A=FF
MOV P1,A ;CONFIGURE P1 INPUT PORT
BACK: MOV A,P1 ;GET X
MOVC A,@A+DPTR ;GET X SQAURE FROM TABLE
MOV P2,A ;ISSUE IT TO P2
SJMP BACK ;KEEP DOING IT

ORG 300H
XSQR_TABLE: DB 0,1,4,9,16,25,36,49,64,81
END
```

1.1.2.23 Example 5-10 Write a program to toggle P1 a total of 200 times. Use RAM location 32H to hold your counter value instead of registers R0 - R7 Solution:

```
MOV P1,#55H ;P1=55H
MOV A, P1
MOV 32H,#200 ;load counter value into RAM loc 32H
LOP1: CPL A ;toggle P1
MOV P1, A
ACALL DELAY
DJNZ 32H,LOP1 ;repeat 200 times

;———————this is the delay subroutine——————
DELAY: MOV R5,#0FFH ;R5=255 (FF in hex), counter
AGAIN: DJNZ R5,AGAIN ;stay here until R5 become 0
RET ;return to caller (when R5 =0)
```

1.1.2.24 Example 5-24 A switch is connected to pin P1.7. Write a program to check the status of the switch and make the following decision. (a) If SW = 0, send '0' to P2 (b) If SW = 1, send '1' to P2 Solution:

```
SW EQU P1.7
MYDATA EQU P2
HERE: MOV C,SW
JC OVER
MOV MYDATA,#'0'
SJMP HERE
OVER: MOV MYDATA,#'1'
SJMP HERE
END
```

1.1.2.25 Example 5-27 Assume that the on-chip ROM has a message. Write a program to copy it from code space into the upper memory space starting at address 80H. Also, as you place a byte in upper RAM, give a copy to P0. Solution:

```
ORG 0
MOV DPTR,#MYDATA
MOV R1,#80H ;access the upper memory
B1: CLR A
MOVC A,@A+DPTR ;copy from code ROM
MOV @R1,A ;store in upper memory
MOV P0,A ;give a copy to P0
JZ EXIT ;exit if last byte
INC DPTR ;increment DPTR
INC R1 ;increment R1
SJMP B1 ;repeat until last byte
EXIT: SJMP $ ;stay here when finished


;——————————————
ORG 300H
MYDATA: DB "The Promise of World Peace",0
END
```

1.1.2.26 Assume that RAM locations 40 - 44H have the following values. Write a program to find the sum of the values. At the end of the program, register A should contain the low byte and R7 the high byte. 40 = (7D), 41 = (EB), 42 = (C5), 43 = (5B), 44 = (30) Solution:

```
MOV R0,#40H ;load pointer
MOV R2,#5 ;load counter
CLR A ;A=0
```

```
MOV R7,A ;clear R7
AGAIN: ADD A,@R0 ;add the byte ptr to by R0
JNC NEXT ;if CY=0 don't add carry
INC R7 ;keep track of carry
NEXT: INC R0 ;increment pointer
DJNZ R2,AGAIN ;repeat until R2 is zero
```

1.1.2.27 Write a program to add two 16-bit numbers. Place the sum in R7 and R6; R6 should have the lower byte. Solution:

```
CLR C ;make CY=0
MOV A, #0E7H ;load the low byte now A=E7H
ADD A, #8DH ;add the low byte
MOV R6, A ;save the low byte sum in R6
MOV A, #3CH ;load the high byte
ADDC A, #3BH ;add with the carry
MOV R7, A ;save the high byte sum
```

1.1.2.28 Assume that 5 BCD data items are stored in RAM locations starting at 40H, as shown below. Write a program to find the sum of all the numbers. The result must be in BCD. 40=(71), 41=(11), 42=(65), 43=(59), 44=(37) Solution:

```
MOV R0,#40H ;Load pointer
MOV R2,#5 ;Load counter
CLR A ;A=0
MOV R7,A ;Clear R7
AGAIN: ADD A,@R0 ;add the byte pointer to by R0
DA A ;adjust for BCD
JNC NEXT ;if CY=0 don't accumulate carry
INC R7 ;keep track of carries
NEXT: INC R0 ;increment pointer
DJNZ R2,AGAIN ;repeat until R2 is 0
```

```
MOV A,#29H ;A=29H, packed BCD
MOV R2,A ;keep a copy of BCD data
ANL A,#0FH ;mask the upper nibble (A=09)
ORL A,#30H ;make it an ASCII, A=39H('9')
MOV R6,A ;save it
MOV A,R2 ;A=29H, get the original data
ANL A,#0F0H ;mask the lower nibble
RR A ;rotate right
RR A ;rotate right
RR A ;rotate right
RR A ;rotate right
ORL A,#30H ;A=32H, ASCII char. '2'
MOV R2,A ;save ASCII char in R2
```

```
CLR P2.3 ;Clear P2.3
MOV TMOD,#01 ;Timer 0, 16-bitmode
HERE: MOV TL0,#3EH ;TL0=3Eh, the low byte
MOV TH0,#0B8H ;TH0=B8H, the high byte
SETB P2.3 ;SET high timer 0
SETB TR0 ;Start the timer 0
AGAIN: JNB TF0,AGAIN ;Monitor timer flag 0
CLR TR0 ;Stop the timer 0
CLR TF0 ;Clear TF0 for next round
CLR P2.3
```

Solution: (a) (FFFFH - B83E + 1) = 47C2H = 18370 in decimal and 18370 _ 1.085 us = 19.93145 ms
(b) Since TH - TL = B83EH = 47166 (in decimal) we have 65536 - 47166 = 18370. This means that the timer counts from B38EH to FFFF. This plus Rolling over to 0 goes through a total of 18370 clock cycles, where each clock is 1.085 us in duration. Therefore, we have 18370 _ 1.085 us = 19.93145 ms as the width of the pulse.

1.1.2.31 Example 9-8 Modify TL and TH in Example 9-7 to get the largest time delay possible. Find the delay in ms. In your calculation, exclude the overhead due to the instructions in the loop. Solution:

To get the largest delay we make TL and TH both 0. This will count up from 0000 to FFFFH and then roll over to zero.

```
CLR P2.3 ;Clear P2.3
MOV TMOD,#01 ;Timer 0, 16−bitmode
HERE: MOV TL0,#0 ;TL0=0, the low byte
MOV TH0,#0 ;TH0=0, the high byte
SETB P2.3 ;SET high P2.3
SETB TR0 ;Start timer 0
AGAIN: JNB TF0,AGAIN ;Monitor timer flag 0
CLR TR0 ;Stop the timer 0
CLR TF0 ;Clear timer 0 flag
CLR P2.3
```

Making TH and TL both zero means that the timer will count from 0000 to FFFF, and then roll over to raise the TF flag. As a result, it goes through a total Of 65536 states. Therefore, we have delay = (65536 - 0) * 1.085 us = 71.1065ms.

```
MOV TMOD,#10;Timer 1, mod 1 (16−bitmode)
AGAIN: MOV TL1,#34H ;TL1=34H, low byte of timer
MOV TH1,#76H ;TH1=76H, high byte timer
SETB TR1 ;start the timer 1
BACK: JNB TF1,BACK ;till timer rolls over
CLR TR1 ;stop the timer 1
CPL P1.5 ;comp. p1. to get hi, lo
CLR TF1 ;clear timer flag 1
SJMP AGAIN ;is not auto−reload
```

Solution: Since FFFFH - 7634H = 89CBH + 1 = 89CCH and 89CCH = 35276 clock count and 35276 * 1.085 us = 38.274 ms for half of the square wave. The frequency = 13.064Hz. Also notice that the high portion and low portion of the square wave pulse are equal. In the above calculation, the overhead due to all the instruction in the loop is not included.

1.1.2.33 Example 9-10 Assume that XTAL = 11.0592 MHz. What value do we need to load the timer's register if we want to have a time delay of 5 ms (milliseconds)? Show the program for timer 0 to create a pulse width of 5 ms on P2.3. Solution:

Since XTAL = 11.0592 MHz, the counter counts up every 1.085 us. This means that out of many 1.085 us intervals we must make a 5 ms pulse. To get that, we divide one by the other. We need 5 ms / 1.085 us = 4608 clocks. To Achieve that we need to load into TL and TH the value 65536 - 4608 = EE00H. Therefore, we have TH = EE and TL = 00.

```
CLR P2.3 ;Clear P2.3
MOV TMOD,#01 ;Timer 0, 16−bitmode
HERE: MOV TL0,#0 ;TL0=0, the low byte
MOV TH0,#0EEH ;TH0=EE, the high byte
SETB P2.3 ;SET high P2.3
SETB TR0 ;Start timer 0
AGAIN: JNB TF0,AGAIN ;Monitor timer flag 0
CLR TR0 ;Stop the timer 0
CLR TF0 ;Clear timer 0 flag
```

1.1.2.34 Example 9-11 Assume that XTAL = 11.0592 MHz, write a program to generate a square wave of 2 kHz frequency on pin P1.5. **Solution:** This is similar to Example 9-10, except that we must toggle the bit to generate the square wave. Look at the following steps.

- (a) T = 1 / f = 1 / 2 kHz = 500 us the period of square wave.
- (b) 1 / 2 of it for the high and low portion of the pulse is 250 us.
- (c) 250 us / 1.085 us = 230 and 65536 - 230 = 65306 which in hex is FF1AH.
- (d) TL = 1A and TH = FF, all in hex. The program is as follow.

```
MOV TMOD,#01 ;Timer 0, 16-bitmode
AGAIN: MOV TL1,#1AH ;TL1=1A, low byte of timer
MOV TH1,#0FFH ;TH1=FF, the high byte
SETB TR1 ;Start timer 1
BACK: JNB TF1,BACK ;until timer rolls over
CLR TR1 ;Stop the timer 1
CLR P1.5 ;Clear timer flag 1
CLR TF1 ;Clear timer 1 flag
SJMP AGAIN ;Reload timer
```

1.1.2.35 Example 9-12 Assume XTAL = 11.0592 MHz, write a program to generate a square wave of 50 kHz frequency on pin P2.3. **Solution:** Look at the following steps.

- (a) T = 1 / 50 = 20 ms, the period of square wave.
- (b) 1 / 2 of it for the high and low portion of the pulse is 10 ms.
- (c) 10 ms / 1.085 us = 9216 and 65536 - 9216 = 56320 in decimal, and in hex it is DC00H.
- (d) TL = 00 and TH = DC (hex).

```
MOV TMOD,#10H ;Timer 1, mod 1
AGAIN: MOV TL1,#00 ;TL1=00,low byte of timer
MOV TH1,#0DCH ;TH1=DC, the high byte
SETB TR1 ;Start timer 1
BACK: JNB TF1,BACK ;until timer rolls over
CLR TR1 ;Stop the timer 1
CLR P2.3 ;Comp. p2.3 to get hi, lo
SJMP AGAIN ;Reload timer, mode 1 isn't auto-reload
```

```
MOV TMOD,#20H ;T1/8-bit/auto reload
MOV TH1,#5 ;TH1 = 5
SETB TR1 ;start the timer 1
BACK: JNB TF1,BACK ;till timer rolls over
CPL P1.0 ;P1.0 to hi, lo
CLR TF1 ;clear Timer 1 flag
SJMP BACK ;mode 2 is auto-reload
```

Solution: First notice the target address of SJMP. In mode 2 we do not need to reload TH since it is auto-reload. Now (256 - 05) _ 1.085 us = 251 _ 1.085 us = 272.33 us is the high portion of the pulse. Since it is a 50% duty cycle square wave, the period T is twice that; as a result T = 2 * 272.33 us = 544.67 us and the frequency = 1.83597 kHz

1.1.2.37 Example 9-15 Find the frequency of a square wave generated on pin P1.0. **Solution:**

```
MOV TMOD,#2H ;Timer 0, mod 2 (8-bit, auto reload)
MOV TH0,#0
AGAIN: MOV R5,#250 ;multiple delay count
ACALL DELAY
CPL P1.0
SJMP AGAIN
```

```
DELAY: SETB TR0 ;start the timer 0
BACK: JNB TF0,BACK ;stay timer rolls over
CLR TR0 ;stop timer
CLR TF0 ;clear TF for next round
DJNZ R5,DELAY
RET ;T = 2 ( 250 * 256 * 1.085 us ) = 138.88ms, and frequency = 72 Hz
```

1.1.2.38 Example 9-18 Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL1 count on P2, which connects to 8 LEDs. Solution:

```
MOV TMOD,#01100000B ;counter 1, mode 2, C/T=1 external pulses
MOV TH1,#0 ;clear TH1
SETB P3.5 ;make T1 input
AGAIN: SETB TR1 ;start the counter
BACK: MOV A,TL1 ;get copy of TL
MOV P2,A ;display it on port 2
JNB TF1,Back ;keep doing, if TF = 0
CLR TR1 ;stop the counter 1
CLR TF1 ;make TF=0
SJMP AGAIN ;keep doing it
```

1.1.2.39 Write a program for the 8051 to transfer letter ‘A’ serially at 4800 baud, continuously. Solution:

```
MOV TMOD,#20H ;timer 1,mode 2(auto reload)
MOV TH1,#-6 ;4800 baud rate
MOV SCON,#50H ;8-bit, 1 stop, REN enabled
SETB TR1 ;start timer 1
AGAIN: MOV SBUF,#"A" ;letter 'A' to transfer
HERE: JNB TI,HERE ;wait for the last bit
CLR TI ;clear TI for next char
SJMP AGAIN ;keep sending A
```

1.1.2.40 Write a program for the 8051 to transfer ‘YES’ serially at 9600 baud, 8-bit data, 1 stop bit, do this continuously Solution:

```
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
AGAIN: MOV A,#"Y" ;transfer 'Y'
ACALL TRANS
MOV A,#"E" ;transfer 'E'
ACALL TRANS
MOV A,#"S" ;transfer 'S'
ACALL TRANS
SJMP AGAIN ;keep doing it
;serial data transfer subroutine
TRANS: MOV SBUF,A ;load SBUF
HERE: JNB TI,HERE ;wait for the last bit
CLR TI ;get ready for next byte
RET
```

1.1.2.41 Write a program for the 8051 to receive bytes of data serially, and put them in P1, set the baud rate at 4800, 8-bit data, and 1 stop bit Solution:

```
MOV TMOD,#20H ;timer 1,mode 2(auto reload)
MOV TH1,#-6 ;4800 baud rate
MOV SCON,#50H ;8-bit, 1 stop, REN enabled
```

```
SETB TR1 ;start timer 1
HERE: JNB RI,HERE ;wait for char to come in
MOV A,SBUF ;saving incoming byte in A
MOV P1,A ;send to port 1
CLR RI ;get ready to receive next byte
SJMP HERE ;keep getting data
```

1.1.2.42 Example 10-5 Assume that the 8051 serial port is connected to the COM port of IBM PC, and on the PC, we are using the terminal.exe program to send and receive data serially. P1 and P2 of the 8051 are connected to LEDs and switches, respectively. Write an 8051 program to: (a) send to PC the message “We Are Ready”. (b) receive any data send by PC and put it on LEDs connected to P1, and (c) get data on switches connected to P2 and send it to PC serially. The program should perform part (a) once, but parts (b) and (c) continuously, use 4800 baud rate. Solution:

```
ORG 0
MOV P2,#0FFH ;make P2 an input port
MOV TMOD,#20H ;timer 1, mode 2
MOV TH1,#0FAH ;4800 baud rate
MOV SCON,#50H ;8-bit, 1 stop, REN enabled
SETB TR1 ;start timer 1
MOV DPTR,#MYDATA ;load pointer for message
H_1: CLR A
MOVC A,@A+DPTR ;get the character
JZ B_1 ;if last character get out
ACALL SEND ;otherwise call transfer
INC DPTR ;next one
SJMP H_1 ;stay in loop
B_1: MOV a,P2 ;read data on P2
ACALL SEND ;transfer it serially
ACALL RECV ;get the serial data
MOV P1,A ;display it on LEDs
SJMP B_1 ;stay in loop indefinitely

;----serial data transfer. ACC has the data------
SEND: MOV SBUF,A ;load the data
H_2: JNB TI,H_2 ;stay here until last bit gone
CLR TI ;get ready for next char
RET ;return to caller

;----Receive data serially in ACC-----------------
RECV: JNB RI,RECV ;wait here for char
MOV A,SBUF ;save it in ACC
CLR RI ;get ready for next char
RET ;return to caller

;------The message----------------
MYDATA: DB "We Are Ready",0
END
```

1.1.2.43 Example 10-6 Assume that XTAL = 11.0592 MHz for the following program, state (a) what this program does, (b) compute the frequency used by timer 1 to set the baud rate, and (c) find the baud rate of the data transfer. Solution:

- (a) This program transfers ASCII letter B (01000010 binary) continuously
- (b) With XTAL = 11.0592 MHz and SMOD = 1 in the above program,
- we have: 11.0592 / 12 = 921.6 kHz
- machine cycle frequency. 921.6 / 16 = 57,600 Hz
- frequency used by timer 1 to set the baud rate.57600 / 3 = 19,200, the baud rate.

```
MOV A,PCON ;A=PCON
MOV ACC.7 ;make D7=1
MOV PCON,A ;SMOD=1, double baud rate with same XTAL freq.
MOV TMOD,#20H ;timer 1, mode 2
MOV TH1,-3 ;19200 (57600/3 =19200)
MOV SCON,#50H ;8-bit data, 1 stop bit, RI enabled
SETB TR1 ;start timer 1
MOV A,#'B' ;transfer letter B
A_1: CLR TI ;make sure TI=0
MOV SBUF,A ;transfer it
H_1: JNB TI,H_1 ;stay here until the last bit is gone
SJMP A_1 ;keep sending 'B' again
```

1.1.2.44 Example 10-10 Write a program to send the message “The Earth is but One Country” to serial port. Assume a SW is connected to pin P1.2. Monitor its status and set the baud rate as follows: SW = 0, 4800 baud rate, SW = 1, 9600 baud rate Assume XTAL = 11.0592 MHz, 8-bit data, and 1 stop bit. Solution:

```
SW BIT P1.2
ORG 0H ;starting position
MAIN:
MOV TMOD,#20H
MOV TH1,#-6 ;4800 baud rate (default)
MOV SCON,#50H
SETB TR1
SETB SW ;make SW an input
 S1: JNB SW,SLOWSP ;check SW status
MOV A,PCON ;read PCON
SETB ACC.7 ;set SMOD high for 9600
MOV PCON,A ;write PCON
SJMP OVER ;send message

SLOWSP:
MOV A,PCON ;read PCON
SETB ACC.7 ;set SMOD low for 4800
MOV PCON,A ;write PCON
OVER: MOV DPTR,#MESS1 ;load address to message
FN: CLR A
MOVC A,@A+DPTR ;read value
 JZ S1 ;check for end of line
ACALL SENDCOM ;send value to serial port
INC DPTR ;move to next value
SJMP FN ;repeat
;———————
SENDCOM:
MOV SBUF,A ;place value in buffer
HERE: JNB TI,HERE ;wait until transmitted
CLR TI ;clear
RET ;return
;———————
MESS1: DB "The Earth is but One Country",0
END
```

1.1.2.45 Example 11-2 Write a program that continuously get 8-bit data from P0 and sends it to P1 while simultaneously creating a square wave of 200 us period on pin P2.1. Use timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz. Solution:

```
 ;We will use timer 0 in mode 2 (auto reload). TH0 = 100/1.085 us = 92
 ;--upon wake-up go to main, avoid using
 ;memory allocated to Interrupt Vector Table
```

```
ORG 0000H
LJMP MAIN ;by-pass interrupt vector table

;--ISR for timer 0 to generate square wave
ORG 000BH ;Timer 0 interrupt vector table
CPL P2.1 ;toggle P2.1 pin
RETI ;return from ISR

;--The main program for initialization
ORG 0030H ;after vector table space
MAIN: MOV TMOD,#02H ;Timer 0, mode 2
MOV P0,#0FFH ;make P0 an input port
MOV TH0,#-92 ;TH0=A4H for -92
MOV IE,#82H ;IE=10000010 (bin) enable Timer 0
SETB TR0 ;Start Timer 0
BACK: MOV A,P0 ;get data from P0
MOV P1,A ;issue it to P1
SJMP BACK ;keep doing it loop unless interrupted by TF0
END
```

1.1.2.46 Example 11-3 Rewrite Example 11-2 to create a square wave that has a high portion of 1085 us and a low portion of 15 us. Assume XTAL=11.0592MHz. Use timer 1.

Solution: Since 1085 us is 1000 * 1.085 we need to use mode 1 of timer 1. upon wake-up go to main, avoid using memory allocated to Interrupt Vector Table

```
ORG 0000H
LJMP MAIN ;by-pass int. vector table
;--ISR for timer 1 to generate square wave
ORG 001BH ;Timer 1 int. vector table
LJMP ISR_T1 ;jump to ISR

;--The main program for initialization
ORG 0030H ;after vector table space
MAIN: MOV TMOD,#10H ;Timer 1, mode 1
MOV P0,#0FFH ;make P0 an input port
MOV TL1,#018H ;TL1=18 low byte of -1000
MOV TH1,#0FCH ;TH1=FC high byte of -1000
MOV IE,#88H ;10001000 enable Timer 1 int
SETB TR1 ;Start Timer 1
BACK: MOV A,P0 ;get data from P0
MOV P1,A ;issue it to P1
SJMP BACK ;keep doing it
;Timer 1 ISR. Must be reloaded, not auto-reload
ISR_T1: CLR TR1 ;stop Timer 1
MOV R2,#4 ; 2MC
CLR P2.1 ;P2.1=0, start of low portion
HERE: DJNZ R2,HERE ;4x2 machine cycle 8MC
MOV TL1,#18H ;load T1 low byte value 2MC
MOV TH1,#0FCH;load T1 high byte value 2MC
SETB TR1 ;starts timer1 1MC
SETB P2.1 ;P2.1=1,back to high 1MC
RETI ;return to main
END
```

1.1.2.47 Example 11-5 Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED. The LED is connected to P1.3 and is normally off. When it is turned on it should stay on for a fraction of a second. As long as the switch is pressed low, the LED should stay on. Solution:

```
ORG 0000H
LJMP MAIN ;by-pass interrupt vector table
```

```
;--ISR for INT1 to turn on LED
ORG 0013H ;INT1 ISR
SETB P1.3 ;turn on LED
MOV R3,#255
BACK: DJNZ R3,BACK ;keep LED on for a while
CLR P1.3 ;turn off the LED
RETI ;return from ISR
;--MAIN program for initialization
ORG 30H
MAIN: MOV IE,#10000100B ;enable external INT 1
HERE: SJMP HERE ;stay here until get interrupted
END
```

1.1.2.48 Assume that pin 3.3 (INT1) is connected to a pulse generator, write a program in which the falling edge of the pulse will send a high to P1.3, which is connected to an LED (or buzzer). In other words, the LED is turned on and off at the same rate as the pulses are applied to the INT1 pin. Solution:

```
ORG 0000H
LJMP MAIN
;--ISR for hardware interrupt INT1 to turn on LED
ORG 0013H ;INT1 ISR
SETB P1.3 ;turn on LED
MOV R3,#255
BACK: DJNZ R3,BACK ;keep the buzzer on for a while
CLR P1.3 ;turn off the buzzer
RETI ;return from ISR
;--------MAIN program for initialization
ORG 30H
MAIN: SETB TCON.2 ;make INT1 edge-triggered int.
MOV IE,#10000100B ;enable External INT 1
HERE: SJMP HERE ;stay here until get interrupted
END
```

1.1.2.49 Example 11-8 Write a program in which the 8051 reads data from P1 and writes it to P2 continuously while giving a copy of it to the serial COM port to be transferred serially. Assume that XTAL=11.0592. Set the baud rate at 9600. Solution:

```
ORG 0000H
LJMP MAIN
ORG 23H
LJMP SERIAL ;jump to serial int ISR
ORG 30H
MAIN: MOV P1,#0FFH ;make P1 an input port
MOV TMOD,#20H ;timer 1, auto reload
MOV TH1,#0FDH ;9600 baud rate
MOV SCON,#50H ;8-bit,1 stop, ren enabled
MOV IE,10010000B ;enable serial int.
SETB TR1 ;start timer 1
BACK: MOV A,P1 ;read data from port 1
MOV SBUF,A ;give a copy to SBUF
MOV P2,A ;send it to P2
SJMP BACK ;stay in loop indefinitely

;--------------------SERIAL PORT ISR
ORG 100H
SERIAL: JB TI,TRANS;jump if TI is high
MOV A,SBUF ;otherwise due to receive
CLR RI ;clear RI since CPU doesn't
RETI ;return from ISR
TRANS: CLR TI ;clear TI since CPU doesn't
```

```
RETI ;return from ISR
END
```

1.1.2.50 Example 11-9 Write a program in which the 8051 gets data from P1 and sends it to P2 continuously while incoming data from the serial port is sent to P0. Assume that XTAL=11.0592. Set the baud rata at 9600. Solution:

```
ORG 0000H
LJMP MAIN
ORG 23H
LJMP SERIAL ;jump to serial int ISR
ORG 30H
MAIN: MOV P1,#0FFH ;make P1 an input port
MOV TMOD,#20H ;timer 1, auto reload
MOV TH1,#0FDH ;9600 baud rate
MOV SCON,#50H ;8-bit ,1 stop, ren enabled
MOV IE,10010000B ;enable serial int.
SETB TR1 ;start timer 1
BACK: MOV A,P1 ;read data from port 1
MOV P2,A ;send it to P2
SJMP BACK ;stay in loop indefinitely

;—————————————SERIAL PORT ISR
ORG 100H
SERIAL: JB TI,TRANS;jump if TI is high
MOV A,SBUF ;otherwise due to receive
MOV P0,A ;send incoming data to P0
CLR RI ;clear RI since CPU doesn't
RETI ;return from ISR
TRANS: CLR TI ;clear TI since CPU doesn't
RETI ;return from ISR
END
```

1.1.2.51 Example 11-10 Write a program using interrupts to do the following: (a) Receive data serially and sent it to P0, (b) Have P1 port read and transmitted serially, and a copy given to P2, (c) Make timer 0 generate a square wave of 5kHz frequency on P0.1. Assume that XTAL-11,0592. Set the baud rate at 4800. Solution:

```
ORG 0
LJMP MAIN
ORG 000BH ;ISR for timer 0
CPL P0.1 ;toggle P0.1
RETI ;return from ISR
ORG 23H ;
LJMP SERIAL ;jump to serial interrupt ISR
ORG 30H
MAIN: MOV P1,#0FFH ;make P1 an input port
MOV TMOD,#22H;timer 1,mode 2(auto reload)
MOV TH1,#0F6H;4800 baud rate
MOV SCON,#50H;8-bit, 1 stop, ren enabled
MOV TH0,#-92 ;for 5kHZ wave
MOV IE,10010010B ;enable serial int.
SETB TR1 ;start timer 1
SETB TR0 ;start timer 0
BACK: MOV A,P1 ;read data from port 1
MOV SBUF,A ;give a copy to SBUF
MOV P2,A ;send it to P2
SJMP BACK ;stay in loop indefinitely
;—————————————SERIAL PORT ISR
ORG 100H
SERIAL:JB TI,TRANS;jump if TI is high
```

```
MOV A,SBUF ;otherwise due to receive
MOV P0,A ;send serial data to P0
CLR RI ;clear RI since CPU doesn't
RETI ;return from ISR
TRANS: CLR TI ;clear TI since CPU doesn't
RETI ;return from ISR
END


ORG 0

MOV A,#38H ;INIT. LCD 2 LINES, 5X7 MATRIX
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time

MOV A,#0EH ;display on, cursor on
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time

MOV A,#01 ;clear LCD
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time

MOV A,#06H ;shift cursor right
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time

MOV A,#84H ;cursor at line 1, pos. 4
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time

MOV A,#'N' ;display letter N
ACALL DATAWRT ;call display subroutine
ACALL DELAY ;give LCD some time

MOV A,#'O' ;display letter O
ACALL DATAWRT ;call display subroutine
AGAIN: SJMP AGAIN ;stay here

COMNWRT: ;send command to LCD
    MOV P1,A ;copy reg A to port 1
    CLR P2.0 ;RS=0 for command
    CLR P2.1 ;R/W=0 for write
    SETB P2.2 ;E=1 for high pulse
    CLR P2.2 ;E=0 for H-to-L pulse
    RET
DATAWRT: ;write data to LCD
    MOV P1,A ;copy reg A to port 1
    CLR P2.0 ;RS=0 for command
    CLR P2.1 ;R/W=0 for write
    SETB P2.2 ;E=1 for high pulse
    CLR P2.2 ;E=0 for H-to-L pulse
    RET
DELAY: MOV R3,#50 ;50 or higher for fast CPUs
HERE2: MOV R4,#255 ;R4 = 255
HERE: DJNZ R4,HERE ;stay until R4 becomes 0
    DJNZ R3,HERE2
    RET
    END
```

```
ORG 0
MOV A,#38H ;init. LCD 2 lines ,5x7 matrix
ACALL COMMAND ;issue command
MOV A,#0EH ;LCD on, cursor on
ACALL COMMAND ;issue command
MOV A,#01H ;clear LCD command
ACALL COMMAND ;issue command
MOV A,#06H ;shift cursor right
ACALL COMMAND ;issue command
MOV A,#86H ;cursor: line 1, pos. 6
ACALL COMMAND ;command subroutine
MOV A,#'N' ;display letter N
ACALL DATA_DISPLAY
MOV A,#'O' ;display letter O
ACALL DATA_DISPLAY
HERE:SJMP HERE ;STAY HERE

COMMAND:
    ACALL READY ;is LCD ready?
    MOV P1,A ;issue command code
    CLR P2.0 ;RS=0 for command
    CLR P2.1 ;R/W=0 to write to LCD
    SETB P2.2 ;E=1 for H-to-L pulse
    CLR P2.2 ;E=0,latch in
    RET
DATA_DISPLAY:
    ACALL READY ;is LCD ready?
    MOV P1,A ;issue data
    SETB P2.0 ;RS=1 for data
    CLR P2.1 ;R/W =0 to write to LCD
    SETB P2.2 ;E=1 for H-to-L pulse
    CLR P2.2 ;E=0,latch in
    RET
READY:
    SETB P1.7 ;make P1.7 input port
    CLR P2.0 ;RS=0 access command reg
    SETB P2.1 ;R/W=1 read command reg

;read command reg and check busy flag
BACK:SETB P2.2 ;E=1 for H-to-L pulse
    CLR P2.2 ;E=0 H-to-L pulse
    JB P1.7,BACK ;stay until busy flag=0
    RET
    END
```

```
ORG 0H

MOV A,#38H ;INIT. LCD 2 LINES, 5X7 MATRIX
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time

MOV A,#0EH ;display on, cursor on
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time

MOV A,#01 ;clear LCD
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
```

```
MOV A,#06H ;shift cursor right
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time

MOV A,#84H ;cursor at line 1, pos. 4
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time

MOV A,#'N' ;display letter N
ACALL DATAWRT ;call display subroutine
ACALL DELAY ;give LCD some time

MOV A,#'O' ;display letter O
ACALL DATAWRT ;call display subroutine
AGAIN: SJMP AGAIN ;stay here

COMNWRT: ;send command to LCD
    MOV P1,A ;copy reg A to port 1
    CLR P2.0 ;RS=0 for command
    CLR P2.1 ;R/W=0 for write
    SETB P2.2 ;E=1 for high pulse
    ACALL DELAY ;give LCD some time
    CLR P2.2 ;E=0 for H-to-L pulse
    RET
DATAWRT: ;write data to LCD
    MOV P1,A ;copy reg A to port 1
    SETB P2.0 ;RS=1 for data
    CLR P2.1 ;R/W=0 for write
    SETB P2.2 ;E=1 for high pulse
    ACALL DELAY ;give LCD some time
    CLR P2.2 ;E=0 for H-to-L pulse
    RET
DELAY: MOV R3,#50 ;50 or higher for fast CPUs
HERE2: MOV R4,#255 ;R4 = 255
HERE: DJNZ R4,HERE ;stay until R4 becomes 0
DJNZ R3,HERE2
RET
END
```

```
ORG 0H

MOV A,#38H ;init. LCD 2 lines ,5x7 matrix
ACALL COMMAND ;issue command

MOV A,#0EH ;LCD on, cursor on
ACALL COMMAND ;issue command

MOV A,#01H ;clear LCD command
ACALL COMMAND ;issue command

MOV A,#06H ;shift cursor right
ACALL COMMAND ;issue command

MOV A,#86H ;cursor: line 1, pos. 6
ACALL COMMAND ;command subroutine

MOV A,#'N' ;display letter N
ACALL DATA_DISPLAY
```

```
MOV A,#'O' ;display letter O
ACALL DATA_DISPLAY
HERE:SJMP HERE ;STAY HERE

COMMAND:
    ACALL READY ;is LCD ready?
    MOV P1,A ;issue command code
    CLR P2.0 ;RS=0 for command
    CLR P2.1 ;R/W=0 to write to LCD
    SETB P2.2 ;E=1 for H-to-L pulse
    CLR P2.2 ;E=0,latch in
    RET
DATA_DISPLAY:
    ACALL READY ;is LCD ready?
    MOV P1,A ;issue data
    SETB P2.0 ;RS=1 for data
    CLR P2.1 ;R/W =0 to write to LCD
    SETB P2.2 ;E=1 for H-to-L pulse
    CLR P2.2 ;E=0,latch in
    RET
READY:
    SETB P1.7 ;make P1.7 input port
    CLR P2.0 ;RS=0 access command reg
    SETB P2.1 ;R/W=1 read command reg
    ;read command reg and check busy flag
BACK:SETB P2.2 ;E=1 for H-to-L pulse
CLR P2.2 ;E=0 H-to-L pulse
JB P1.7,BACK ;stay until busy flag=0
RET
END
```

```
ORG 0
MOV DPTR,#MYCOM
C1: CLR A
    MOVC A,@A+DPTR
    ACALL COMNWRT ;call command subroutine
    ACALL DELAY ;give LCD some time
    INC DPTR
    JZ SEND_DAT
    SJMP C1
SEND_DAT:
MOV DPTR,#MYDATA
D1: CLR A
    MOVC A,@A+DPTR
    ACALL DATAWRT ;call command subroutine
    ACALL DELAY ;give LCD some time
    INC DPTR
    JZ AGAIN
    SJMP D1
AGAIN: SJMP AGAIN ;stay here

COMNWRT: ;send command to LCD
    MOV P1,A ;copy reg A to P1
    CLR P2.0 ;RS=0 for command
    CLR P2.1 ;R/W=0 for write
    SETB P2.2 ;E=1 for high pulse
    ACALL DELAY ;give LCD some time
    CLR P2.2 ;E=0 for H-to-L pulse
    RET
```

```
DATAWRT: ;write data to LCD
    MOV P1,A ;copy reg A to port 1
    SETB P2.0 ;RS=1 for data
    CLR P2.1 ;R/W=0 for write
    SETB P2.2 ;E=1 for high pulse
    ACALL DELAY ;give LCD some time
    CLR P2.2 ;E=0 for H-to-L pulse
    RET
DELAY: MOV R3,#250 ;50 or higher for fast CPUs
HERE2: MOV R4,#255 ;R4 = 255
HERE: DJNZ R4,HERE ;stay until R4 becomes 0
DJNZ R3,HERE2
RET

 ;ORG 300H
 ;MYCOM: DB 38H,0EH,01,06,84H,0 ; commands and null
 ;MYDATA: DB "HELLO",0
 ;END



MOV P2,#0FFH ;make P2 an input port
K1: MOV P1,#0 ;ground all rows at once
MOV A,P2 ;read all col
 ;(ensure keys open)
ANL A,00001111B ;masked unused bits
CJNE A,#00001111B,K1 ;till all keys release
K2: ACALL DELAY ;call 20 msec delay
MOV A,P2 ;see if any key is pressed
ANL A,00001111B ;mask unused bits
CJNE A,#00001111B,OVER;key pressed, find row
SJMP K2 ;check till key pressed
OVER: ACALL DELAY ;wait 20 msec debounce time
MOV A,P2 ;check key closure
ANL A,00001111B ;mask unused bits
CJNE A,#00001111B,OVER1;key pressed, find row
SJMP K2 ;if none, keep polling
OVER1: MOV P1, #11111110B ;ground row 0
MOV A,P2 ;read all columns
ANL A,#00001111B ;mask unused bits
CJNE A,#00001111B,ROW_0 ;key row 0, find col.
MOV P1,#11111101B ;ground row 1
MOV A,P2 ;read all columns
ANL A,#00001111B ;mask unused bits
CJNE A,#00001111B,ROW_1 ;key row 1, find col.
MOV P1,#11111011B ;ground row 2
MOV A,P2 ;read all columns
ANL A,#00001111B ;mask unused bits
CJNE A,#00001111B,ROW_2 ;key row 2, find col.
MOV P1,#11110111B ;ground row 3
MOV A,P2 ;read all columns
ANL A,#00001111B ;mask unused bits
CJNE A,#00001111B,ROW_3 ;key row 3, find col.
LJMP K2 ;if none, false input, repeat

ROW_0: MOV DPTR,#KCODE0 ;set DPTR=start of row 0
SJMP FIND ;find col. Key belongs to
ROW_1: MOV DPTR,#KCODE1 ;set DPTR=start of row
SJMP FIND ;find col. Key belongs to
ROW_2: MOV DPTR,#KCODE2 ;set DPTR=start of row 2
SJMP FIND ;find col. Key belongs to
```

```
ROW_3: MOV DPTR,#KCODE3 ;set DPTR=start of row 3
FIND: RRC A ;see if any CY bit low
JNC MATCH ;if zero, get ASCII code
INC DPTR ;point to next col. addr
SJMP FIND ;keep searching
MATCH: CLR A ;set A=0 (match is found)
MOVC A,@A+DPTR ;get ASCII from table
MOV P0,A ;display pressed key
LJMP K1

;ASCII LOOK-UP TABLE FOR EACH ROW
ORG 300H
KCODE0: DB '0','1','2','3' ;ROW 0
KCODE1: DB '4','5','6','7' ;ROW 1
KCODE2: DB '8','9','A','B' ;ROW 2
KCODE3: DB 'C','D','E','F' ;ROW 3

;------------ this is delay subroutine --------------
ORG 400H ;put DELAY at address 300H
DELAY: MOV R5,#0FFH ;R5=255 (FF in hex), counter
AGAIN: DJNZ R5,AGAIN ;stay here until R5 become 0
RET ;return to caller (when R5 =0)

END
```

1.2 Embedded C Programs

1.2.1 Mazidi Book C Programs

```c
#include <reg51.h>
void main(void)
{
    unsigned char z;
    for (z = 0; z <= 255; z++)
        P1 = z;
}
```

```c
#include <reg51.h>
void main(void)
{
    unsigned char mynum[] = "012345ABCD";
    unsigned char z;
    for (z = 0; z <= 10; z++)
        P1 = mynum[z];
}
```

```c
// Toggle P1 forever
#include <reg51.h>
void main(void)
{
    for (;;)
    {
        P1 = 0x55;
        P1 = 0xAA;
    }
}
```

```
//Singed numbers
#include <reg51.h>
void main(void)
{
    char mynum[] = {+1, -1, +2, -2, +3, -3, +4, -4};
    unsigned char z;
    for (z = 0; z <= 8; z++)
        P1 = mynum[z];
}
```

```
#include <reg51.h>
sbit MYBIT = P1 ^ 0;
void main(void)
{
    unsigned int z;
    for (z = 0; z <= 50000; z++)
    {
        MYBIT = 0;
        MYBIT = 1;
    }
}
```

```
//Toggle P1 forever with some delay in between
//"on" and "off"
#include <reg51.h>
void main(void)
{
    unsigned int x;
    for (;;) // repeat forever
    {
        P1 = 0x55;
        for (x = 0; x < 40000; x++)
            ; // delay size
        // unknown
        P1 = 0xAA;
        for (x = 0; x < 40000; x++)
            ;
    }
}
```

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
    while (1) // repeat forever
    {
        P1 = 0x55;
        MSDelay(250);
        P1 = 0xAA;
        MSDelay(250);
    }
}

void MSDelay(unsigned int itime)
{
    unsigned int i, j;
    for (i = 0; i < itime; i++)
        for (j = 0; j < 1275; j++)
            ;
}
```

```
#include <reg51.h>
#define LED P2
void main(void)
{
    P1 = 00; // clear P1
    LED = 0; // clear P2
    for (;;) // repeat forever
    {
        P1++;  // increment P1
        LED++; // increment P2
    }
}
```

```
#include <reg51.h>
void MSDelay(unsigned int);

void main(void)
{
    unsigned char mybyte;
    P1 = 0xFF; // make P1 input port
    while (1)
    {
        mybyte = P1; // get a byte from P1
        MSDelay(500);
        P2 = mybyte; // send it to P2
    }
}

void MSDelay(unsigned int itime)
{
    unsigned int i, j;
    for (i = 0; i < itime; i++)
        for (j = 0; j < 1275; j++)
            ;
}
```

```
#include <reg51.h>
void main(void)
{
    unsigned char mybyte;
    P0 = 0xFF; // make P0 input port
    while (1)
    {
        mybyte = P0; // get a byte from P0
        if (mybyte < 100)
            P1 = mybyte; // send it to P1
        else
            P2 = mybyte; // send it to P2
    }
}
```

```
//Toggling an individual bit
#include <reg51.h>
sbit mybit = P2 ^ 4;
void main(void)
{
    while (1)
    {
        mybit = 1; // turn on P2.4
        mybit = 0; // turn off P2.4
    }
}
```

```
#include <reg51.h>
sbit mybit = P1 ^ 5;
void main(void)
{
    mybit = 1; // make mybit an input
    while (1)
    {
        if (mybit == 1)
            P0 = 0x55;
        else
            P2 = 0xAA;
    }
}
```

```
#include <reg51.h>
void MSDelay(unsigned int);
sbit Dsensor = P1 ^ 1;
sbit Buzzer = P1 ^ 7;

void main(void)
{
    Dsensor = 1; // make P1.1 an input
    while (1)
    {
        while (Dsensor == 1) // while it opens
        {
            Buzzer = 0;
            MSDelay(200);
            Buzzer = 1;
            MSDelay(200);
        }
    }
}
void MSDelay(unsigned int itime)
{
    unsigned int i, j;
    for (i = 0; i < itime; i++)
        for (j = 0; j < 1275; j++)
            ;
}
```

```
#include <reg51.h>
#define LCDData P1 // LCDData declaration
sbit En = P2 ^ 0;  // the enable pin

void main(void)
{
    unsigned char message[] = "The␣Earth␣is␣but␣One␣Country";
    unsigned char z;
    for (z = 0; z < 28; z++) // send 28 characters
    {
        LCDData = message[z];
        En = 1; // a highEn=0; //-to-low pulse to latch data
    }
}
```

```
// Accessing Ports as SFRs using sfr data type
sfr P0 = 0x80;
sfr P1 = 0x90;
sfr P2 = 0xA0;
void MSDelay(unsigned int);

void main(void)
{
    while (1)
    {
        P0 = 0x55;
        P1 = 0x55;
        P2 = 0x55;
        MSDelay(250);
        P0 = 0xAA;
        P1 = 0xAA;
        P2 = 0xAA;
        MSDelay(250);
    }
}

void MSDelay(unsigned int itime)
{
    unsigned int i, j;
    for (i = 0; i < itime; i++)
        for (j = 0; j < 1275; j++)
            ;
}
```

```
#include <reg51.h>
```

```c
sbit MYBIT = 0x95;
void main(void)
{
    unsigned int z;
    for (z = 0; z < 50000; z++)
    {
        MYBIT = 1;
        MYBIT = 0;
    }
}
```

```c
#include <reg51.h>
sbit inbit = P1 ^ 0;
sbit outbit = P2 ^ 7;
bit membit; // use bit to declare bit- addressable memory

void main(void)
{
    while (1)
    {
        membit = inbit;  // get a bit from P1.0
        outbit = membit; // send it to P2.7
    }
}
```

```c
#include <reg51.h>
void main(void)
{
    P0 = 0x35 & 0x0F; // ANDing
    P1 = 0x04 | 0x68; // ORing
    P2 = 0x54 ^ 0x78; // XORing
    P0 = ~0x55;       // inversing
    P1 = 0x9A >> 3;   // shifting right 3
    P2 = 0x77 >> 4;   // shifting right 4
    P0 = 0x6 << 4;    // shifting left 4
}
```

```c
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
    P0 = 0x55;
    P2 = 0x55;
    while (1)
    {
        P0 = ~P0;
        P2 = P2 ^ 0xFF;
        MSDelay(250);
    }
}

void MSDelay(unsigned int itime)
{
    unsigned int i, j;
    for (i = 0; i < itime; i++)
        for (j = 0; j < 1275; j++)
            ;
}
```

```
#include <reg51.h>
sbit inbit = P1 ^ 0;
sbit outbit = P2 ^ 7;
bit membit;

void main(void)
{
    while (1)
    {
        membit = inbit;     // get a bit from P1.0
        outbit = ~membit; // invert it and send it to P2.7
    }
}
```

1.2.1.21 Write an 8051 C program to read the P1.0 and P1.1 bits and issue an ASCII character to P0 according to the following table.

P1.1 P1.0 0 0 send ‘0’ to P0 0 1 send ‘1’ to P0 1 0 send ‘2’ to P0 1 1 send ‘3’ to P0

```
#include <reg51.h>
void main(void)
{
    unsigned char z;
    z = P1;
    z = z & 0x3;
    switch (z)
    {
    case (0):
    {
        P0 = '0';
        break;
    }
    case (1):
    {
        P0 = '1';
        break;
    }
    case (2):
    {
        P0 = '2';
        break;
    }
    case (3):
    {
        P0 = '3';
        break;
    }
    }
}
```

```
#include <reg51.h>
void main(void)
```

```
{
    unsigned char x, y, z;
    unsigned char mybyte = 0x29;
    x = mybyte & 0x0F;
    P1 = x | 0x30;
    y = mybyte & 0xF0;
    y = y >> 4;
    P2 = y | 0x30;
}
```

```
#include <reg51.h>
void main(void)
{
    unsigned char bcdbyte;
    unsigned char w = '4';
    unsigned char z = '7';
    w = w & 0x0F;
    w = w << 4;
    z = z & 0x0F;
    bcdbyte = w | z;
    P1 = bcdbyte;
}
```

```
#include <reg51.h>
void main(void)
{
    unsigned char mydata[] = {0x25, 0x62, 0x3F, 0x52};
    unsigned char sum = 0;
    unsigned char x;
    unsigned char chksumbyte;
    for (x = 0; x < 4; x++)
    {
        P2 = mydata[x];
        sum = sum + mydata[x];
        P1 = sum;
    }
    chksumbyte = ~sum + 1;
    P1 = chksumbyte;
}
```

```
#include <reg51.h>
void main(void)
{
    unsigned char mydata[] = {0x25, 0x62, 0x3F, 0x52, 0xE8};
    unsigned char chksum = 0;
    unsigned char x;
    for (x = 0; x < 5; x++)
        chksum = chksum + mydata[x];
    if (chksum == 0)
        P0 = 'G';
    else
        P0 = 'B';
}
```

```c
#include <reg51.h>
void main(void)
{
    unsigned char x, binbyte, d1, d2, d3;
    binbyte = 0xFD;
    x = binbyte / 10;
    d1 = binbyte % 10;
    d2 = x % 10;
    d3 = x / 10;
    P0 = d1;
    P1 = d2;
    P2 = d3;
}
```

```c
#include <reg51.h>
void main(void)
{
    unsigned char mynum[] = "ABCDEF"; // RAM space
    unsigned char z;
    for (z = 0; z <= 6; z++)
        P1 = mynum[z];
}
```

```c
#include <reg51.h>
void main(void)
{
    unsigned char mydata[100]; // RAM space
    unsigned char x, z = 0;
    for (x = 0; x < 100; x++)
    {
        z--;
        mydata[x] = z;
        P1 = z;
    }
}
```

```c
#include <reg51.h>
void main(void)
{
    code unsigned char mynum[] = "ABCDEF";
    unsigned char z;
    for (z = 0; z <= 6; z++)
        P1 = mynum[z];
}
```

```c
#include <reg51.h>
void main(void)
{
    unsigned char mydata[100]; // RAM space
    unsigned char x, z = 0;
    for (x = 0; x < 100; x++)
    {
            z--;
            mydata[x] = z;
            P1 = z;
        }
}
```

```
#include <reg51.h>
void main(void)
{
        code unsigned char mynum[] = "ABCDEF";
        unsigned char z;
        for (z = 0; z <= 6; z++)
            P1 = mynum[z];
}
```

```
#include <reg51.h>
sbit P1b0 = P1 ^ 0;
sbit regALSB = ACC ^ 0;
void main(void)
{
        unsigned char conbyte = 0x44;
        unsigned char x;
        ACC = conbyte;
        for (x = 0; x < 8; x++)
        {
            P1b0 = regALSB;
            ACC = ACC >> 1;
        }
}
```

```
#include <reg51.h>
sbit P1b0 = P1 ^ 0;
sbit regAMSB = ACC ^ 7;
void main(void)
{
        unsigned char conbyte = 0x44;
        unsigned char x;
        ACC = conbyte;
        for (x = 0; x < 8; x++)
        {
            P1b0 = regAMSB;
            ACC = ACC << 1;
        }
}
```

```
#include <reg51.h>
sbit P1b0 = P1 ^ 0;
sbit ACCMSB = ACC ^ 7;
bit membit;
void main(void)
{
        unsigned char x;
```

```
    for (x = 0; x < 8; x++)
    {
        membit = P1b0;
        ACC = ACC >> 1;
        ACCMSB = membit;
    }
    P2 = ACC;
}
```

```
#include <reg51.h>
sbit P1b0 = P1 ^ 0;
sbit regALSB = ACC ^ 0;
bit membit;

void main(void)
{
    unsigned char x;
    for (x = 0; x < 8; x++)
    {
        membit = P1b0;
        ACC = ACC << 1;
        regALSB = membit;
    }
    P2 = ACC;
}
```

```
#include <reg51.h>
void T0Delay(void);
void main(void)
{
    while (1)
    {
        P1 = 0x55;
        T0Delay();
        P1 = 0xAA;
        T0Delay();
    }
}

void T0Delay()
{
    TMOD = 0x01;
    TL0 = 0x00;
    TH0 = 0x35;
    TR0 = 1;
    while (TF0 == 0)
        ;
    TR0 = 0;
    TF0 = 0;
}
```

```
#include <reg51.h>
void T0M1Delay(void);
sbit mybit = P1 ^ 5;
```

```
void main(void)
{
    while (1)
    {
        mybit = ~mybit;
        T0M1Delay();
    }
}

void T0M1Delay(void)
{
    TMOD = 0x01;
    TL0 = 0xFD;
    TH0 = 0x4B;
    TR0 = 1;
    while (TF0 == 0)
        ;
    TR0 = 0;
    TF0 = 0;
}
```

```
#include <reg51.h>
void T1M1Delay(void);
void main(void)
{
    unsigned char x;
    P2 = 0x55;
    while (1)
    {
        P2 = ~P2;
        for (x = 0; x < 20; x++)
            T1M1Delay();
    }
}

void T1M1Delay(void)
{
    TMOD = 0x10;
    TL1 = 0xFE;
    TH1 = 0xA5;
    TR1 = 1;
    while (TF1 == 0)
        ;
    TR1 = 0;
    TF1 = 0;
}
```

```
#include <reg51.h>
sbit mybit = P1 ^ 5;
sbit SW = P1 ^ 7;
void T0M1Delay(unsigned char);

void main(void)
{
    SW = 1;
    while (1)
    {
        mybit = ~mybit;
        if (SW == 0)
            T0M1Delay(0);
        else
            T0M1Delay(1);
    }
}

void T0M1Delay(unsigned char c)
{
    TMOD = 0x01;
    if (c == 0)
    {
        TL0 = 0x67;
        TH0 = 0xFC;
    }
    else
    {
        TL0 = 0x9A;
        TH0 = 0xFD;
    }
    TR0 = 1;
    while (TF0 == 0)
        ;
    TR0 = 0;
    TF0 = 0;
}
```

```
#include <reg51.h>
void T0M2Delay(void);
sbit mybit = P1 ^ 5;

void main(void)
{
    unsigned char x, y;
    while (1)
    {
        mybit = ~mybit;
        for (x = 0; x < 250; x++)
            for (y = 0; y < 36; y++) // we put 36, not 40
                T0M2Delay();
    }
}

void T0M2Delay(void)
{
    TMOD = 0x02;
    TH0 = -23;
    TR0 = 1;
    while (TF0 == 0)
        ;
    TR0 = 0;
    TF0 = 0;
}
```

```
#include <reg51.h>
```

```
void T1M2Delay(void);
sbit mybit = P2 ^ 7;

void main(void)
{
    unsigned char x;
    while (1)
    {
        mybit = ~mybit;
        T1M2Delay();
    }
}
void T1M2Delay(void)
{
    TMOD = 0x20;
    TH1 = -184;
    TR1 = 1;
    while (TF1 == 0)
        ;
    TR1 = 0;
    TF1 = 0;
}
```

```
#include <reg51.h>
void main(void)
{
    T1 = 1;
    TMOD = 0x60;
    TH1 = 0;
    while (1)
    {
        do
        {
            TR1 = 1;
            P1 = TL1;
        } while (TF1 == 0);
        TR1 = 0;
        TF1 = 0;
    }
}
```

```
#include <reg51.h>
void main(void)
{
    T0 = 1;
    TMOD = 0x05;
    TL0 = 0;
    TH0 = 0;
    while (1)
    {
        do
        {
            TR0 = 1;
            P1 = TL0;
            P2 = TH0;
        } while (TF0 == 0);
        TR0 = 0;
        TF0 = 0;
    }
}
```

```
#include <reg51.h>
void main(void)
{
    TMOD = 0x20; // use Timer 1, mode 2
    TH1 = 0xFA;  // 4800 baud rate
    SCON = 0x50;
    TR1 = 1;
    while (1)
    {
        SBUF = 'A'; // place value in buffer
        while (TI == 0)
            ;
        TI = 0;
    }
}
```

```
#include <reg51.h>
void SerTx(unsigned char);
void main(void)
{
    TMOD = 0x20; // use Timer 1, mode 2
    TH1 = 0xFD;  // 9600 baud rate
    SCON = 0x50;
    TR1 = 1; // start timer
    while (1)
    {
        SerTx('Y');
        SerTx('E');
        SerTx('S');
    }
}

void SerTx(unsigned char x)
{
    SBUF = x; // place value in buffer
    while (TI == 0)
        ; // wait until transmitted
    TI = 0;
}
```

```
#include <reg51.h>
void main(void)
{
    unsigned char mybyte;
    TMOD = 0x20; // use Timer 1, mode 2
    TH1 = 0xFA;  // 4800 baud rate
    SCON = 0x50;
    TR1 = 1; // start timer
    while (1)
    { // repeat forever
```

```
        while (RI == 0)
            ;           // wait to receive
        mybyte = SBUF; // save value
        P1 = mybyte;   // Write value to port
        RI = 0;
    }
}
```

```
#include <reg51.h>
sbit MYSW = P2 ^ 0; // input switch
void main(void)
{
    unsigned char z;
    unsigned char Mess1[] = "Normal␣Speed";
    unsigned char Mess2[] = "High␣Speed";
    TMOD = 0x20; // use Timer 1, mode 2
    TH1 = 0xFF;      // 28800 for normal
    SCON = 0x50;
    TR1 = 1; // start timer
    if (MYSW == 0)
    {
        for (z = 0; z < 12; z++)
        {
            SBUF = Mess1[z]; // place value in buffer
            while (TI == 0)
                ; // wait for transmit
            TI = 0;
        }
    }
    else
    {
        PCON = PCON | 0x80; // for high speed of 56K
        for (z = 0; z < 10; z++)
        {
            SBUF = Mess2[z]; // place value in buffer
            while (TI == 0)
                ; // wait for transmit
            TI = 0;
        }
    }
}
```

```
#include <reg51.h>
sfr SBUF1 = 0xC1;
sfr SCON1 = 0xC0;
sbit TI1 = 0xC1;

void main(void)
{
    TMOD = 0x20; // use Timer 1, mode 2
    TH1 = 0xFA;  // 4800 baud rate
    SCON = 0x50; // use 2nd serial port SCON1
    TR1 = 1;     // start timer
    while (1)
    {
        SBUF1 = 'A'; // use 2nd serial port SBUF1
        while (TI1 == 0)
```

```
            ; // wait for transmit
        TI1 = 0;
    }
}
```

```
#include <reg51.h>
sfr SBUF1 = 0xC1;
sfr SCON1 = 0xC0;
sbit RI1 = 0xC0;
void main(void)
{
    unsigned char mybyte;
    TMOD = 0x20;   // use Timer 1, mode 2
    TH1 = 0xFD;    // 9600 baud rate
    SCON1 = 0x50;  // use 2nd serial port SCON1
    TR1 = 1;       // start timer
    while (1)
    {
        while (RI1 == 0)
            ;              // monitor RI1
        mybyte = SBUF1;    // use SBUF1
        P2 = mybyte;       // place value on port
        RI1 = 0;
    }
}
```

```
// We will use timer 0 mode 2 (auto-reload). One half of the period is 100
    ↪ us. 100/1.085 us = 92, and TH0 = 256 - 92 = 164 or A4H
#include <reg51.h>
sbit SW = P1 ^ 7;
sbit IND = P1 ^ 0;
sbit WAVE = P2 ^ 5;

void timer0(void) interrupt 1
{
    WAVE = ~WAVE; // toggle pin
}

void main()
{
    SW = 1; // make switch input
    TMOD = 0x02;
    TH0 = 0xA4; // TH0=-92
    IE = 0x82;     // enable interrupt for timer 0
    while (1)
    {
        IND = SW; // send switch to LED
    }
}
```

```
#include <reg51.h>
sbit WAVE = P0 ^ 1;

void timer0() interrupt 1
```

```
{
    WAVE = ~WAVE; // toggle pin
}

void serial0() interrupt 4
{
    if (TI == 1)
    {
        TI = 0; // clear interrupt
    }
    else
    {
        P0 = SBUF; // put value on pins
        RI = 0;    // clear interrupt
    }
}

void main()
{
    unsigned char x;
    P1 = 0xFF; // make P1 an input
    TMOD = 0x22;
    TH1 = 0xF6; // 4800 baud rate
    SCON = 0x50;
    TH0 = 0xA4; // 5 kHz has T=200us
    IE = 0x92;  // enable interrupts
    TR1 = 1;    // start timer 1
    TR0 = 1;    // start timer 0
    while (1)
    {
        x = P1;   // read value from pins
        SBUF = x; // put value in buffer
        P2 = x;   // Write value to pins
    }
}
```

```
#include <reg51.h>
sbit WAVE = P2 ^ 1;
unsigned char cnt;
void timer0() interrupt 1
{
    WAVE = ~WAVE; // toggle pin
}
void timer1() interrupt 3
{
    cnt++;    // increment counter
    P0 = cnt; // display value on pins
}

void main()
{
    cnt = 0; // set counter to 0
    TMOD = 0x42;
    TH0 = 0x46; // 10 KHz
    IE = 0x86;  // enable interrupts
    TR0 = 1;    // start timer 0
    while (1)
        ; // wait until interrupted
}
```

```
#include <reg51.h>
sfr ldata = 0x90; // P1=LCD data pins
sbit rs = P2 ^ 0;
sbit rw = P2 ^ 1;
sbit en = P2 ^ 2;
sbit busy = P1 ^ 7;

void MSDelay(unsigned int itime)
{
    unsigned int i, j;
    for (i = 0; i < itime; i++)
        for (j = 0; j < 1275; j++)
            ;
}

void lcdready()
{
    busy = 1; // make the busy pin at input
    rs = 0;
    rw = 1;
    while (busy == 1)
    {           // wait here for busy flag
        en = 0; // strobe the enable pin
        MSDelay(1);
        en = 1;
    }
}

void lcdcmd(unsigned char value)
{
    lcdready();     // check the LCD busy flag
    ldata = value; // put the value on the pins
    rs = 0;
    rw = 0;
    en = 1; // strobe the enable pin
    MSDelay(1);
    en = 0;
    return;
}

void lcddata(unsigned char value)
{
    lcdready();     // check the LCD busy flag
    ldata = value; // put the value on the pins
    rs = 1;
    rw = 0;
    en = 1; // strobe the enable pin
    MSDelay(1);
    en = 0;
    return;
}

void main()
{
    lcdcmd(0x38);
    lcdcmd(0x0E);
    lcdcmd(0x01);
    lcdcmd(0x06);
    lcdcmd(0x86); // line 1, position 6
    lcddata('M');
    lcddata('D');
    lcddata('E');
}
```