CSE-3103: Microprocessor and Microcontroller

Dept. of Computer Science and Engineering
University of Dhaka

Prof. Sazzad M.S. Imran, PhD
Dept. of Electrical and Electronic Engineering
sazzadmsi.webnode.com

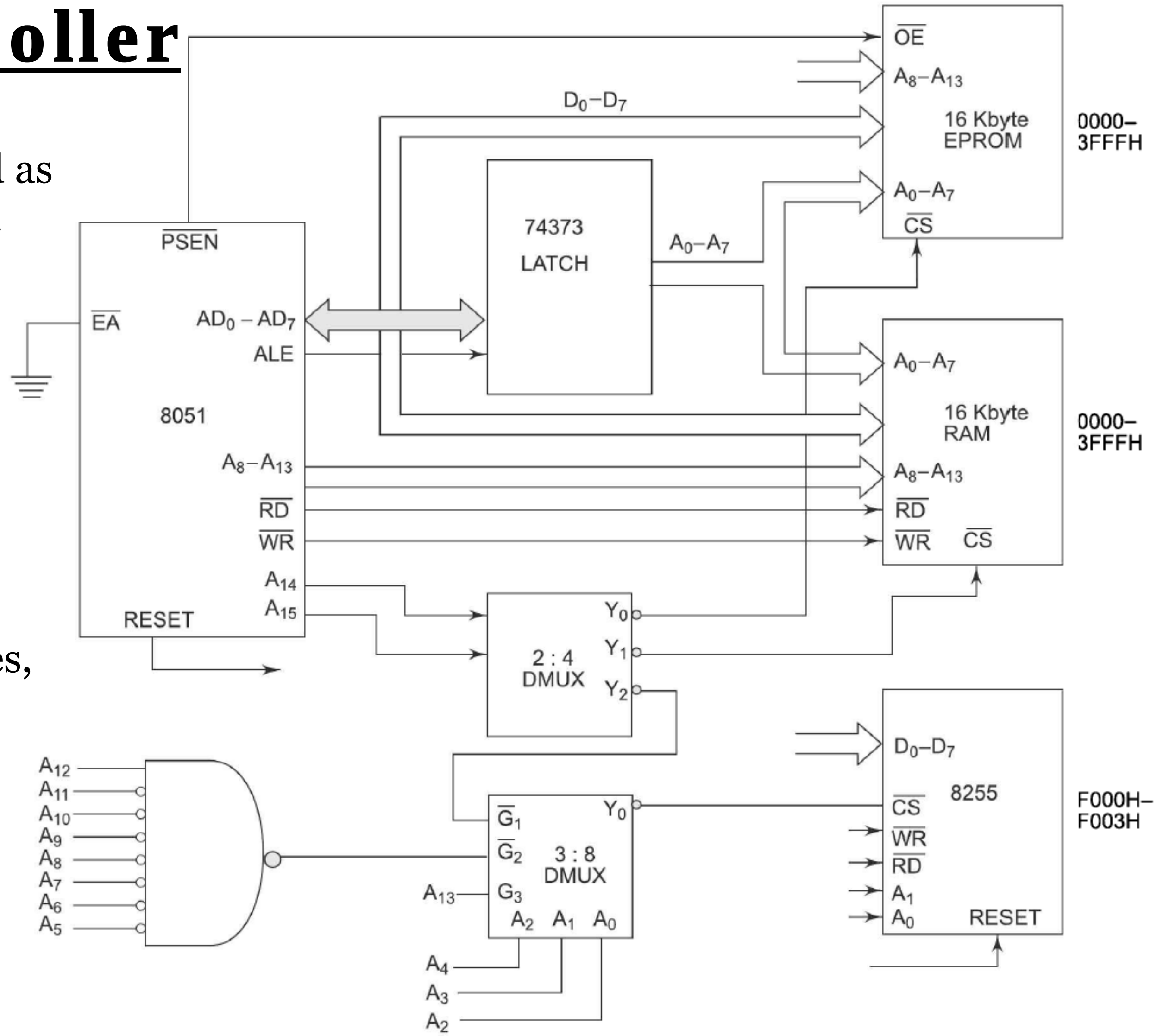
8051 Microcontroller

External I/O Interfacing →
external I/O devices are interfaced as
external memory-mapped devices.

devices are treated as
external memory locations.

devices consume
external memory addresses.

System →
external RAM memory of 16K bytes,
external ROM of 16K bytes,
8255 PPI is interfaced externally.



8051 Microcontroller

Interrupts →

5 sources of interrupts.

$\overline{INT_0}$ and $\overline{INT_1}$ →

2 external interrupt inputs.

programmed with bits IT_0 and IT_1 in register TCON.

processed internally by flags IE_0 and IE_1.

Process = external interrupt occurs,
IE_0 or IE_1 is set,
CPU jumps to respective interrupt vector.

programmed as edge-sensitive →

flags are automatically cleared after
control is transferred to respective vector.
must remain high for at least 1 machine cycle and
low for at least 1 machine cycle.

programmed as level-sensitive →

flags are controlled by external interrupts sources themselves.
must remain high for at least 2 machine cycles.

both timers are used in timer or counter mode.

counter mode →

counts external pulses at T_0 or T_1 pin.

8051 Microcontroller

Interrupts →

5 sources of interrupts.

$\overline{INT_0}$ and $\overline{INT_1}$ →

timer mode →

oscillator clock is divided by prescaler 1/32,

then given to timer.

clock frequency for timer = (oscillator frequency)/32.

timer is up-counter (0000H – FFFFH),

count = FFFFH → generates interrupt.

operated in 4 different modes →

set by TMOD register.

Mode 0: 13-bit Timer (0000H – 1FFFH)

Mode 1: 16-bit Timer (0000H – FFFFH)

Mode 2: 8-bit Auto-Reload (00H – FFH)

Mode 3: Split Timer Mode

(Timer 0 = 2 separate 8-bit timers,

Timer 1 = reserved)

8051 Microcontroller

Interrupts →

- 5 sources of interrupts.
- Timer 0 and timer 1 interrupts →
  - interrupt sources are generated by TF_0 and TF_1 bits of register TCON,
  - each timer increments based on selected timer/counter mode.
  - timer reaches its maximum count,
  - rollover takes place in respective timer registers →
    - TF_0 and TF_1 are set.
    - interrupts are generated →
      - control is transferred to respective ISR,
      - respective flags are automatically cleared.

- Serial port interrupt →
  - byte has been received and/or transmitted.
  - bits RI and/or TI is set →
    - interrupt is generated.
    - control is transferred to interrupt service routine.
  - neither of flags is automatically cleared.

8051 Microcontroller

Interrupts →

5 sources of interrupts.

Serial port interrupt →

in ISR →

decides which one caused interrupt,

corresponding flag is cleared using software.

additional interrupts →

single step interrupts to be generated with software.

interrupts are enabled using special function register IE.

interrupt structure provides 2 levels of interrupt priorities.

each interrupt source is programmed to have 1 of 2 levels.

priorities are programmed using special function register IP.

8051 Microcontroller

Stack →

stack operations are 8-bit wide,
PUSH = 1 byte of data is stored on to stack,
POP = 1 byte of data is retrieved from stack.
internal 16-bit address push or pop →

operation is implemented byte by byte,
lower byte first followed by higher byte.

SP register →

8-bit register,
initialized to internal RAM address 07H after reset.
points to stack top.
stack top is always assumed to be preoccupied.

8051 Microcontroller

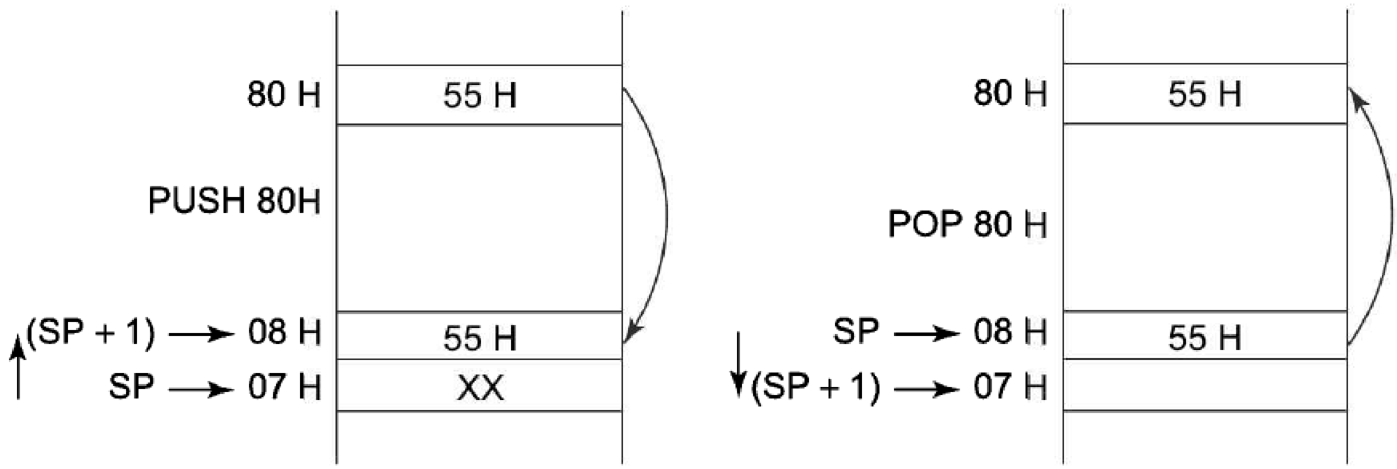
Stack →

PUSH →

1) increment stack (SP) by 1,
2) store 8-bit content of 8-bit address specified in instruction, to address pointed by SP.

POP →

1) store content of top of stack pointed by SP register, to 8-bit memory specified in instruction.
2) decrement SP by 1.



8051 Microcontroller

Problem 1: Write an assembly language program to find whether a given byte is available in the given sequence or not. If it is available, write FF in R3. Otherwise write 00 in R3.

Solution:

```
// A program for finding a number in array of numbers stored in data memory
// for example in memory locations {20H, 21H, 22H, 23H} the contents are [15, 04, 06, 45]
ORG     0000H                   // Sets program start address at location 0000H
        MOV     R0, #20H        // 20H is starting address of array
        MOV     R3, #00H        // in R3 register we monitor search
                                // if R3 = 00H means number not found
                                // if R3 = FFH means number is found
        MOV     R1, #04H        // This is counter, no of elements are 4 in array
AGAIN:
        MOV     A, @R0          // A is stored with contents of memory location (@r0)
        CJNE    A, #45H, NEXT           // If number do not match with 45H then
                                        // jump to NEXT label
        MOV     R3, #0FFH       // Store FFH in R3 indicating 45 is present in array
        SJMP    DONE            // Skip NEXT label instructions and jump to DONE
```

8051 Microcontroller

Problem 1: Write an assembly language program to find whether a given byte is available in the given sequence or not. If it is available, write FF in R3. Otherwise write 00 in R3.

Solution:

```
AGAIN:
        MOV     A, @R0          // A is stored with contents of memory location (@r0)
        CJNE    A, #45H, NEXT           // If number do not match with 45H then
                                        // jump to NEXT label
        MOV     R3, #0FFH       // Store FFH in R3 indicating 45 is present in array
        SJMP    DONE            // Skip NEXT label instructions and jump to DONE
NEXT:
        INC     R0              // Memory address incremented
        DJNZ    R1, AGAIN       // Decrement counter and jump to AGAIN until R1 = 0
DONE:
        END
```

8051 Microcontroller

Problem 2: Write an assembly language program to count the number of 1s and 0s in a given 8-bit number.

Solution:

```
// Program to compute number of 1s and 0s in 8 bit number
// logic: initialize R1 and R2 with 00H; initialize R3 as a counter
// clear carry flag (C) and rotate A along with carry
// if C = 1, increment R1, else increment R2 and decrement the counter
// if counter = 0, store the contents of r1 and r2 and end the program
ORG     0000H                   // Sets program start address at location 0000H.
        MOV     A, #05H         // Number is 05H i.e. 00000101
        MOV     R1, #00H        // Counter for 1s
        MOV     R2, #00H        // Counter for 0s
        MOV     R3, #08H        // Counter for total number of bits
        CLR     C
UP:
        RRC     A               // Rotate right through carry
        JNC     DOWN            // If carry is not present goto label down
        INC     R1              // Increment R1 counter
        SJMP    EXIT            // Skip DOWN label instructions and jump to EXIT
```

8051 Microcontroller

Problem 2: Write an assembly language program to count the number of 1s and 0s in a given 8-bit number.

Solution:

```
DOWN:
        INC     R2              // Increment count of 0s
EXIT:
        DJNZ    R3, UP          // Decrement counter and jump to UP label until R3 = 0
                                // Checking for end of 8 bits
        MOV     R0, #40H        // Initialize R0 to a free internal RAM address
        MOV     A, R1
        MOV     @R0, A          // Store number of 1s at memory location 40H
        INC     R0              // Move to next RAM location
        MOV     A, R2
        MOV     @R0, A          // Store number of 0s at memory location 41H
        END
```

8051 Microcontroller

Problem 3: Write an assembly language program to compute *x* to the power *n* where both *x* and *n* are 8-bit numbers given by user and the result should not be more than 16 bits.

Solution:

```
// logic of this program: x is multiplied to itself n-1 times.
ORG     0000H                       // Set program start address at 0000H
        MOV     A, #02H             // This is base x
        MOV     B, #03H             // This is exponent n
        MOV     R0, B
        MOV     R1, A
        MOV     R2, #01H            // Initialize lower byte of result to 1
LOOP1:
        MOV     A, R2               // Load current lower byte of result into accumulator
        MOV     B, R1               // Load base x into B
        MUL     AB                  // Multiplication; result: LB in A, HB in B
        DEC     R0                  // Decrementing counter
        MOV     R2, A               // Store lower byte of result to R2
        CJNE    R0, #00H, LOOP1     // If R0 ≠ 00H, jump to LOOP1 (Compare R0)
        MOV     A, R2               // Result is stored in accumulator
        END
```

8051 Microcontroller

Problem 4: Write an assembly language program to perform addition of two 2×2 matrices.
Solution:

```
// Let the Contents of A be [5, 6; 7, 8] stored at memory locations {20H, 21H, 22H, 23H}.
// Let contents of B are [3, 2; 1, 0] stored in Memory locations {30H, 31H, 32H, 33H}.
// The result of the addition is to be stored in matrix C = A+B in Memory locations
// {20H, 21H, 22H, 23H}, i.e. by overwriting the addresses of Matrix A.
// R0 handles A and R1 handles B.
ORG     0000H                   // Set program start address at 0000H
        MOV     R0, #20H        // Starting address of A in R0
        MOV     R1, #30H        // Starting address of B in R1
        MOV     R3, #00H        // Clearing R3
        MOV     R4, #04H        // Counter = 4 (no. of elements)
AGAIN:
        MOV     A, @R0          // Contents of A matrix stored in A
        MOV     R3, A           // Temporarily stored in R3
        MOV     A, @R1          // Contents of B matrix stored in A
        ADD     A, R3           // Added with R3
        MOV     @R0, A          // Result of addition is written at addresses of Matrix A
```

8051 Microcontroller

Problem 4: Write an assembly language program to perform addition of two 2×2 matrices.

Solution:

```
AGAIN:
        MOV     A, @R0          // Contents of A matrix stored in A
        MOV     R3, A           // Temporarily stored in R3
        MOV     A, @R1          // Contents of B matrix stored in A
        ADD     A, R3           // Added with R3
        MOV     @R0, A          // Result of addition is written at addresses of Matrix A
        DEC     R4              // Counter is decremented
        INC     R0              // Memory location incremented
        INC     R1              // Memory location incremented
        CJNE    R4, #00H, AGAIN         // until counter becomes 0
                                        // (all values added?) if not, goto label AGAIN
        END
```

8051 Microcontroller

Problem 5: Write an assembly language program for finding transpose of a 2×2 matrix.
Solution:

```
//a program to find transpose of a matrix stored in data memory of 8051
//content of matrix is a = [10, 20; 30, 50] (2 rows and 2 columns [A00, A01; A10, A11])
//stored sequentially at 20H, 21H, 22H, 23H; store result at 30H, 31H, 32H, 33H
ORG     0000H
        MOV     R0, #20H
        MOV     R1, #30H
        MOV     A, @R0
        MOV     @R1, A          // A00 to B00 stored
        INC     R0
        INC     R1
        INC     R1
        MOV     A, @R0
        MOV     @R1, A          // A01 to B10 stored
        INC     R0
        DEC     R1
        MOV     A, @R0
        MOV     @R1, A          // A10 to B01 stored
```

8051 Microcontroller

Problem 5: Write an assembly language program for finding transpose of a 2×2 matrix.

Solution:

```
MOV   A, @R0
MOV   @R1, A        // A01 to B10 stored
INC   R0
DEC   R1
MOV   A, @R0
MOV   @R1, A        // A10 to B01 stored
INC   R0
INC   R1
INC   R1
MOV   A, @R0
MOV   @R1, A        // A11 to B11 stored
                    // Content of transpose matrix is B = [10, 30; 20, 50]
                    // (2 rows and 2 columns [B00, B01; B10, B11])
END
```

8051 Microcontroller

Problem 6: Write an assembly language program for computing square root of an 8 bit number.

Solution:

// logic: We can calculate square root of a number by using iterative technique

// $x_{j+1} = \frac{x_j + \frac{N}{x_j}}{2}$ where x_{j+1} gives us square root of a number N. As j increments,

// we get the result of the next iteration; when $x_{j+1} = x_j$, it is the value of the square root.

```
        N1      EQU 40H         // address of number whose square root is to be calculated
        N       EQU 41H         // Location where answer is to be stored
ORG     0000H
        MOV     N1, #0FH        // Store number whose root is to be calculated
        MOV     B, #01H         // Initial guess x0 = 1
        MOV     R1, B           // Initialize square root to 01
        LCALL   TRY             // Call subroutine for next iteration
        JMP     STOP
TRY:
        MOV     A, N1           // A = N
        MOV     B, R1           // B = xj
```

8051 Microcontroller

Problem 6: Write an assembly language program for computing square root of an 8 bit number.

Solution:

```
TRY:
        MOV     A, N1              // A = N
        MOV     B, R1              // B = x_j
        DIV     AB                 // Calculate N/x_j; result: A = N/x_j, B = remainder
        ADD     A, R1              // Calculate x_j + N/x_j
        CLR     C
        RRC     A                  // Divide by 2, A = (x_j+N/x_j)/2
        CLR      C
        MOV     R2, A              // R2 = x_j+1, new approximation
        SUBB    A, R1              // Get x_{j+1} − x_j
        CJNE    A, #00H, NOTEQ          // If the difference is not zero, continue iteration
        SJMP    OVER               // If the difference is zero, square has been computed in R2
```

8051 Microcontroller

Problem 6: Write an assembly language program for computing square root of an 8 bit number.

Solution:

```
        SJMP    OVER            // If the difference is zero, square has been computed in R2
NOTEQ:
        JC      OVER            // If result < previous guess, done
                                // (If A = x_{j+1} – x_j < 00H in CJNE C = 1)
                                // Otherwise continue with next iteration
        MOV     A, R2
        MOV     R1, A           // Replace x_j with x_{j+1}, R1 = x_{j+1}
        SJMP    TRY             // Go for the next iteration
OVER:
        MOV     N, R2           // Store answer
        RET
STOP:
        NOP                     // safe placeholder instruction that does nothing
        END
```