Microcontrollers Notes for B.E. IV Sem ECE/ETE Students

Visvesvaraya Technological University (VTU), Belagavi
by
Saneesh Cleatus Thundiyil
Associate Professor,
Department of Electronics and Communication Engineering,
BMS Institute of Technology and Management
Bengaluru – 64

SYLLABUS

MODULE I

8051 Microcontroller: Microprocessor Vs Microcontroller, Embedded Systems, Embedded Microcontrollers, 8051 Architecture- Registers, Pin diagram, I/O ports functions, Internal Memory organization. External Memory (ROM & RAM) interfacing.

MODULE II

8051 Instruction Set: Addressing Modes, Data Transfer instructions, Arithmetic instructions, Logical instructions, Branch instructions, Bit manipulation instructions. Simple Assembly language program examples (without loops) to use these instructions.

MODULE III

8051 Stack, I/O Port Interfacing and Programming: 8051 Stack, Stack and Subroutine instructions. Assembly language program examples on subroutine and involving loops. Interfacing simple switch and LED to I/O ports to switch on/off LED with respect to switch status.

MODULE IV

8051 Timers and Serial Port: 8051 Timers and Counters – Operation and Assembly language programming to generate a pulse using Mode-1 and a square wave using Mode-2 on a port pin. 8051 Serial Communication- Basics of Serial Data Communication, RS-232 standard, 9 pin RS232 signals, Simple Serial Port programming in Assembly and C to transmit a message and to receive data serially.

MODULE V

8051 Interrupts and Interfacing Applications: 8051 Interrupts. 8051 Assembly language programming to generate an external interrupt using a switch, 8051 C programming to generate a square waveform on a port pin using a Timer interrupt. Interfacing 8051 to ADC-0804, DAC, LCD and Stepper motor and their 8051 Assembly language interfacing programming.

Text Books:

1. "The 8051 Microcontroller and Embedded Systems – using assembly and C", Muhammad Ali Mazidi and Janice Gillespie Mazidi and Rollin D. McKinlay; PHI, 2006 / Pearson, 2006.
2. "The 8051 Microcontroller", Kenneth J. Ayala, 3rd Edition, Thomson/Cengage Learning.

Reference Books:

1. "The 8051 Microcontroller Based Embedded Systems", Manish K Patel, McGraw Hill, 2014, ISBN: 978-93-329-0125-4.
2. "Microcontrollers: Architecture, Programming, Interfacing and System Design", Raj Kamal, Pearson Education, 2005.

MODULE - 2

Syllabus: 8051 Instruction Set: Addressing Modes, Data Transfer instructions, Arithmetic instructions, Logical instructions, Branch instructions, Bit manipulation instructions. Simple Assembly language program examples (without loops) to use these instructions.

2.1 INSTRUCTION SYNTAX.

General syntax for 8051 assembly language is as follows.

```
LABEL: OPCODE OPERAND ;COMMENT
```

LABEL : (THIS IS NOT NECESSARY UNLESS THAT SPECIFIC LINE HAS TO BE ADDRESSED). The label is a symbolic address for the instruction. When the program is assembled, the label will be given specific address in which that instruction is stored. Unless that specific line of instruction is needed by a branching instruction in the program, it is not necessary to label that line.

OPCODE: Opcode is the symbolic representation of the operation. The assembler converts the opcode to a unique binary code (machine language).

OPERAND: While opcode specifies what operation to perform, operand specifies where to perform that action. The operand field generally contains the source and destination of the data. In some cases only source or destination will be available instead of both. The operand will be either address of the data, or data itself.

COMMENT: Always comment will begin with ; or // symbol. To improve the program quality, programmer may always use comments in the program.

2.2 ADDRESSING MODES

Various methods of accessing the data are called addressing modes.

8051 addressing modes are classified as follows.

1. **Immediate addressing.**
2. **Register addressing.**
3. **Direct addressing.**
4. **Register Indirect addressing.**
5. **Indexed addressing.**
6. *Relative addressing.*
7. *Absolute addressing.*
8. *Long addressing.*
9. *Bit inherent addressing.*
10. *Bit direct addressing.*

1. **Immediate addressing.**

In this addressing mode the data is provided as a part of instruction itself. In other words data immediately follows the instruction.

Eg. MOV A,#30H
ADD A, #83 # Symbol indicates the data is immediate.

2. **Register addressing.**

In this addressing mode the register will hold the data. One of the eight general registers (R0 to R7) can be used and specified as the operand.

Eg. MOV A,R0
ADD A,R6

R0 – R7 will be selected from the current selection of register bank. The default register bank will be bank 0.

3. **Direct addressing**

There are two ways to access the internal memory. Using direct address and indirect address. Using direct addressing mode we can not only address the internal memory but SFRs also. In direct addressing, an 8 bit internal data memory address is specified as part of the instruction and hence, it can specify the address only in the range of 00H to FFH. In this addressing mode, data is obtained directly from the memory.

Eg. MOV A,60h
ADD A,30h

4. **Register Indirect addressing**

The indirect addressing mode uses a register to hold the actual address that will be used in data movement. Registers R0 and R1 and DPTR are the only registers that can be used as data pointers. Indirect addressing cannot be used to refer to SFR registers. Both R0 and R1 can hold 8 bit address and DPTR can hold 16 bit address.

Eg. MOV A,@R0
ADD A,@R1
MOVX A,@DPTR

5. **Indexed addressing.**

In indexed addressing, either the program counter (PC), or the data pointer (DTPR)—is used to hold the base address, and the A is used to hold the offset address. Adding the value of the base address to the value of the offset address forms the effective address. Indexed addressing is used with JMP or MOVC instructions. Look up tables are easily implemented with the help of index addressing.

Eg. MOVC A, @A+DPTR // copies the contents of memory location pointed by the sum of the accumulator A and the DPTR into accumulator A.
MOVC A, @A+PC // copies the contents of memory location pointed by the sum of the accumulator A and the program counter into accumulator A.

6. *Relative Addressing.*

Relative addressing is used only with conditional jump instructions. The relative address, (offset), is an 8 bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8 bit signed offset value gives an address range of +127 to —128 locations. The jump destination is usually specified using a label and the assembler calculates the jump offset accordingly. The advantage of relative addressing is that the program code is easy to relocate and the address is relative to position in the memory.

Eg. SJMP LOOP1
JC BACK

7. *Absolute addressing*

Absolute addressing is used only by the AJMP (Absolute Jump) and ACALL (Absolute Call) instructions. These are 2 bytes instructions. The absolute addressing mode specifies the lowest 11 bit of the memory address as part of the instruction.

The upper 5 bit of the destination address are the upper 5 bit of the current program counter. Hence, absolute addressing allows branching only within the current 2 Kbyte page of the program memory.

```
Eg.     AJMP LOOP1
        ACALL LOOP2
```

8. *Long Addressing*

The long addressing mode is used with the instructions LJMP and LCALL. These are 3 byte instructions. The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64 Kbyte code memory space.

```
Eg.     LJMP FINISH
        LCALL DELAY
```

9. *Bit Inherent Addressing*

In this addressing, the address of the flag which contains the operand, is implied in the opcode of the instruction.

```
Eg.     CLR C       ;       Clears the carry flag to 0
```

10. *Bit Direct Addressing*

In this addressing mode the direct address of the bit is specified in the instruction. The RAM space 20H to 2FH and most of the special function registers are bit addressable. Bit address values are between 00H to 7FH.

```
Eg.     CLR 07h     ;       Clears the bit 7 of 20h RAM space
        SETB 07H    ;       Sets the bit 7 of 20H RAM space.
```

2.3 INSTRUCTION SET.

Instruction Timings

The 8051 internal operations and external read/write operations are controlled by the oscillator clock.
T-state, Machine cycle and Instruction cycle are terms used in instruction timings.
T-state is defined as one subdivision of the operation performed in one clock period. The terms 'T-state' and 'clock period' are often used synonymously.
Machine cycle is defined as 12 oscillator periods. A machine cycle consists of six states and each state lasts for two oscillator periods. An instruction takes one to four machine cycles to execute an instruction. Instruction cycle is defined as the time required for completing the execution of an instruction. The 8051 instruction cycle consists of one to four machine cycles.

Eg. If 8051 microcontroller is operated with 12 MHz oscillator, find the execution time for the following four instructions.

1. ADD A, 45H
2. SUBB A, #55H
3. MOV DPTR, #2000H
4. MUL AB

Since the oscillator frequency is 12 MHz, the clock period is, Clock period = 1/12 MHz = 0.08333 µS.
Time for 1 machine cycle = 0.08333 µS x 12 =1 µS.

	Instruction	No. of machine cycles	Execution time
1.	ADD A, 45H	1	1 µs
2.	SUBB A, #55H	2	2 µs
3.	MOV DPTR, #2000H	2	2 µs
4.	MUL AB	4	4 µs

2.3.1 8051 Instructions

The instructions of 8051 can be broadly classified under the following headings.

1. **Data transfer instructions**
2. **Arithmetic instructions**
3. **Logical instructions**
4. **Branch instructions**
5. **Bit manipulation instructions**
6. Subroutine instructions (will study in module 3)

DATA TRANSFER INSTRUCTIONS.

In this group, the instructions perform data transfer operations of the following types.

a. Move the contents of a register Rn to A
   - *i.* MOV A,R2
   - *ii.* MOV A,R7

b. Move the contents of a register A to Rn
   - *i.* MOV R4,A
   - *ii.* MOV R1,A

c. Move an immediate 8 bit data to register A or to Rn or to a memory location(direct or indirect)
   - *i.* MOV A, #45H
   - *ii.* MOV R6, #51H
   - *iii.* MOV 30H, #44H
   - *iv.* MOV @R0, #0E8H
   - *v.* MOV DPTR, #0F5A2H
   - *vi.* MOV DPTR, #5467H

d. Move the contents of a memory location to A or A to a memory location using direct and indirect addressing
   - *i.* MOV A, 65H
   - *ii.* MOV A, @R0
   - *iii.* MOV 45H, A
   - *iv.* MOV @R1, A

e. Move the contents of a memory location to Rn or Rn to a memory location using direct addressing
   - *i.* MOV R3, 65H
   - *ii.* MOV 45H, R2

f. Move the contents of memory location to another memory location using direct and indirect addressing
   - *i.* MOV 47H, 65H
   - *ii.* MOV 45H, @R0

g. Move the contents of an external memory to A or A to an external memory
   - *i.* MOVX A,@R1
   - *ii.* MOVX @R0,A
   - *iii.* MOVX A,@DPTR
   - *iv.* MOVX@DPTR,A

h. Move the contents of program memory to A
   i. MOVC A, @A+PC
   ii. MOVC A, @A+DPTR

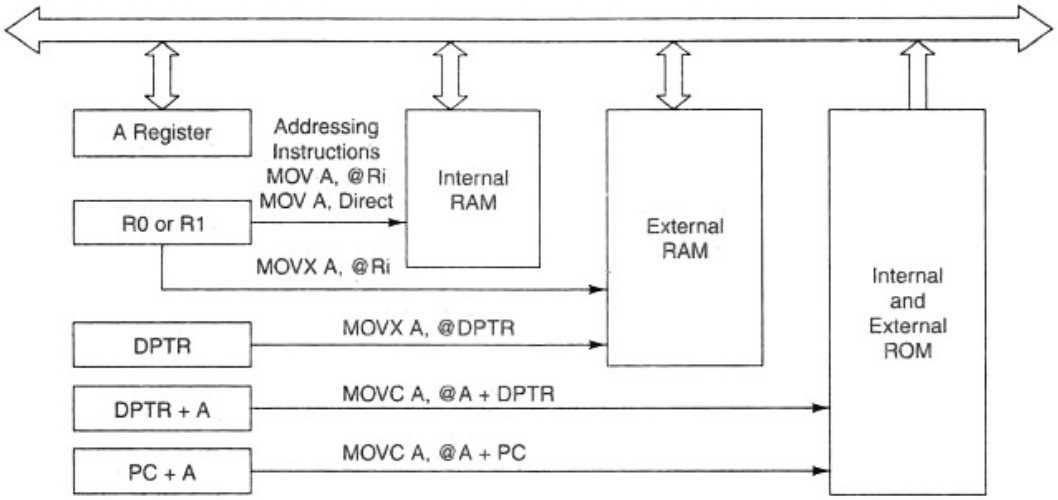


FIG. Addressing Using MOV, MOVX and MOVC

i. Push and Pop instructions

```
                    [SP]=07 //CONTENT OF SP IS 07 (DEFAULT VALUE)
MOV R6, #25H        [R6]=25H        //CONTENT OF R6 IS 25H
MOV R1, #12H        [R1]=12H        //CONTENT OF R1 IS 12H
MOV R4, #0F3H       [R4]=F3H        //CONTENT OF R4 IS F3H

PUSH 6              [SP]=08 [08]=[06]=25H   //CONTENT OF 08 IS 25H
PUSH 1              [SP]=09 [09]=[01]=12H   //CONTENT OF 09 IS 12H
PUSH 4              [SP]=0A [0A]=[04]=F3H   //CONTENT OF 0A IS F3H

POP 6               [06]=[0A]=F3H   [SP]=09 //CONTENT OF 06 IS F3H
POP 1               [01]=[09]=12H   [SP]=08 //CONTENT OF 01 IS 12H
POP 4               [04]=[08]=25H   [SP]=07 //CONTENT OF 04 IS 25H
```

j. Exchange instructions

The content of source ie., register, direct memory or indirect memory will be exchanged with the contents of destination ie., accumulator.

   i. XCH A,R3
   ii. XCH A,@R1
   iii. XCH A,54h

k. Exchange digit. Exchange the lower order nibble of Accumulator (A0-A3) with lower order nibble of the internal RAM location which is indirectly addressed by the register.
   i. XCHD A,@R1
   ii. XCHD A,@R0

ARITHMETIC INSTRUCTIONS.

The 8051 can perform addition, subtraction. Multiplication and division operations on 8 bit numbers.

Addition

In this group, we have instructions to

i. Add the contents of A with immediate data with or without carry.
   i. ADD A, #45H
   ii. ADDC A, #OB4H

ii. Add the contents of A with register Rn with or without carry.
   i. ADD A, R5
   ii. ADDC A, R2

iii. Add the contents of A with contents of memory with or without carry using direct and indirect addressing
   i. ADD A, 51H
   ii. ADDC A, 75H
   iii. ADD A, @R1
   iv. ADDC A, @R0

CY AC and OV flags will be affected by this operation.

Subtraction

In this group, we have instructions to

i. Subtract the contents of A with immediate data with or without carry.
   i. SUBB A, #45H
   ii. SUBB A, #OB4H

ii. Subtract the contents of A with register Rn with or without carry.
   i. SUBB A, R5
   ii. SUBB A, R2

iii. Subtract the contents of A with contents of memory with or without carry using direct and indirect addressing
   i. SUBB A, 51H
   ii. SUBB A, 75H
   iii. SUBB A, @R1
   iv. SUBB A, @R0

CY AC and OV flags will be affected by this operation.

Multiplication

MUL AB. This instruction multiplies two 8 bit unsigned numbers which are stored in A and B register. After multiplication the lower byte of the result will be stored in accumulator and higher byte of result will be stored in B register.

```
Eg.   MOV A,#45H        ;[A]=45H
      MOV B,#0F5H       ;[B]=F5H
      MUL AB            ;[A] x [B] = 45 x F5 = 4209
                        ;[A]=09H, [B]=42H
```

Division

DIV AB. This instruction divides the 8 bit unsigned number which is stored in A by the 8 bit unsigned number which is stored in B register. After division the result will be stored in accumulator and

remainder will be stored in B register.

```
Eg.   MOV A,#45H        ;[A]=0E8H
      MOV B,#0F5H       ;[B]=1BH
      DIV AB            ;[A] / [B] = E8 /1B = 08 H with remainder  10H
                        ;[A] = 08H, [B]=10H
```

DA A (Decimal Adjust After Addition).

When two BCD numbers are added, the answer is a non-BCD number. To get the result in BCD, we use DA A instruction after the addition. DA A works as follows.

- If lower nibble is greater than 9 or auxiliary carry is 1, 6 is added to lower nibble.
- If upper nibble is greater than 9 or carry is 1, 6 is added to upper nibble.

```
Eg 1:  MOV A,#23H
       MOV R1,#55H
       ADD A,R1        // [A]=78
       DA A            // [A]=78        no changes in the accumulator after da a

Eg 2:  MOV A,#53H
       MOV R1,#58H
       ADD A,R1        // [A]=ABh
       DA A            // [A]=11, C=1 . ANSWER IS 111. Accumulator data is changed after DA A
```

Increment: increments the operand by one.

INC A INC Rn INC DIRECT INC @Ri INC DPTR

INC increments the value of source by 1. If the initial value of register is FFh, incrementing the value will cause it to reset to 0. The Carry Flag is not set when the value "rolls over" from 255 to 0.

In the case of "INC DPTR", the value two-byte unsigned integer value of DPTR is incremented. If the initial value of DPTR is FFFFh, incrementing the value will cause it to reset to 0.

Decrement: decrements the operand by one.

DEC A DEC Rn DEC DIRECT DEC @Ri

DEC decrements the value of source by 1. If the initial value of is 0, decrementing the value will cause it to reset to FFh. The Carry Flag is not set when the value "rolls over" from 0 to FFh.

LOGICAL INSTRUCTIONS

Logical AND

ANL destination, source: ANL does a bitwise "AND" operation between source and destination, leaving the resulting value in destination. The value in source is not affected. "AND" instruction logically AND the bits of source and destination.

```
ANL A,#DATA     ANL A, Rn
ANL A,DIRECT    ANL A,@Ri
ANL DIRECT,A    ANL DIRECT, #DATA
```

Logical OR

ORL destination, source: ORL does a bitwise "OR" operation between source and destination, leaving the resulting value in destination. The value in source is not affected. " OR " instruction logically OR the bits of source and destination.

```
ORL A,#DATA     ORL A, Rn
```

ORL A,DIRECT ORL A,@Ri
ORL DIRECT,A ORL DIRECT, #DATA

Logical Ex-OR

XRL destination, source: XRL does a bitwise "EX-OR" operation between source and destination, leaving the resulting value in destination. The value in source is not affected. " XRL " instruction logically EX-OR the bits of source and destination.
XRL A,#DATA XRL A,Rn
XRL A,DIRECT XRL A,@Ri
XRL DIRECT,A XRL DIRECT, #DATA

Logical NOT

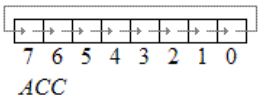
CPL complements operand, leaving the result in operand. If operand is a single bit then the state of the bit will be reversed. If operand is the Accumulator then all the bits in the Accumulator will be reversed.

CPL A, CPL C, CPL bit address

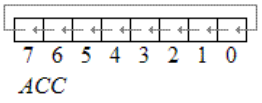
SWAP A – Swap the upper nibble and lower nibble of A.

Rotate Instructions

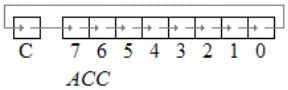
RR A
This instruction is rotate right the accumulator. Its operation is illustrated below. Each bit is shifted one location to the right, with bit 0 going to bit 7.



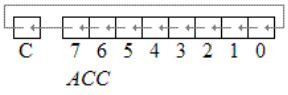
RL A
Rotate left the accumulator. Each bit is shifted one location to the left, with bit 7 going to bit 0



RRC A
Rotate right through the carry. Each bit is shifted one location to the right, with bit 0 going into the carry bit in the PSW, while the carry was at goes into bit 7



RLC A
Rotate left through the carry. Each bit is shifted one location to the left, with bit 7 going into the carry bit in the PSW, while the carry goes into bit 0.

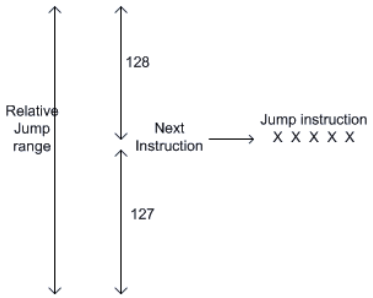


BRANCH INSTRUCTIONS

Jump and Call Program Range
There are 3 types of jump instructions. They are:-

1. Relative Jump
2. Short Absolute Jump
3. Long Absolute Jump

Relative Jump
Jump that replaces the PC (program counter) content with a new address that is greater than (the address following the jump instruction by 127 or less) or less than (the address following the jump by 128 or less) is called a relative jump. Schematically, the relative jump can be shown as follows: -



The advantages of the relative jump are as follows:-

1. Only 1 byte of jump address needs to be specified in the 2's complement form, ie. For jumping ahead, the range is 0 to 127 and for jumping back, the range is -1 to -128.
2. Specifying only one byte reduces the size of the instruction and speeds up program execution.
3. The program with relative jumps can be relocated without reassembling to generate absolute jump addresses.

Disadvantages of the absolute jump: -

1. Short jump range (-128 to 127 from the instruction following the jump instruction)

Instructions that use Relative Jump

SJMP <relative address>; this is unconditional jump

The remaining relative jumps are conditional jumps

JC <relative address>
JNC <relative address>
JB bit, <relative address>
JNB bit, <relative address>
JBC bit, <relative address>
CJNE <destination byte>, <source byte>, <relative address>
DJNZ <byte>, <relative address>
JZ <relative address>
JNZ <relative address>

Short Absolute Jump
In this case only 11bits of the absolute jump address are needed. The absolute jump address is calculated in the following manner.
In 8051, 64 kbyte of program memory space is divided into 32 pages of 2 kbyte each. The hexadecimal addresses of the pages are given as follows:-

Page (Hex)	Address (Hex)
00	0000 - 07FF
01	0800 - 0FFF
02	1000 - 17FF
03	1800 - 1FFF
.	
.	
1E	F000 - F7FF
1F	F800 - FFFF

It can be seen that the upper 5bits of the program counter (PC) hold the page number and the lower 11bits of the PC hold the address within that page. Thus, an absolute address is formed by taking page numbers of the instruction (from the program counter) following the jump and attaching the specified 11bits to it to form the 16-bit address.

Advantage: The instruction length becomes 2 bytes.

Example of short absolute jump: -

```
ACALL <address 11>
AJMP  <address 11>
```

Long Absolute Jump/Call

Applications that need to access the entire program memory from 0000H to FFFFH use long absolute jump. Since the absolute address has to be specified in the op-code, the instruction length is 3 bytes (except for JMP @ A+DPTR). This jump is not re-locatable.

Example: -

```
LCALL <address 16>
LJMP  <address 16>
JMP @A+DPTR
```

Another classification of jump instructions is

1. Unconditional Jump
2. Conditional Jump

1. The unconditional jump is a jump in which control is transferred unconditionally to the target location.
    a. LJMP (long jump). This is a 3-byte instruction. First byte is the op-code and second and third bytes represent the 16-bit target address which is any memory location from 0000 to FFFFH
    eg: LJMP 3000H
    b. AJMP: this causes unconditional branch to the indicated address, by loading the 11 bit address to 0 -10 bits of the program counter. The destination must be therefore within the same 2K blocks.
    c. SJMP (short jump). This is a 2-byte instruction. First byte is the op-code and second byte is the relative target address, 00 to FFH (forward +127 and backward -128 bytes from the current PC value). To calculate the target address of a short jump, the second byte is added to the PC value which is address of the instruction immediately below the jump.

2. Conditional Jump instructions.

JBC	Jump if bit = 1 and clear bit
JNB	Jump if bit = 0
JB	Jump if bit = 1
JNC	Jump if CY = 0
JC	Jump if CY = 1
CJNE reg,#data	Jump if byte ≠ #data
CJNE A,byte	Jump if A ≠ byte
DJNZ	Decrement and Jump if A ≠ 0
JNZ	Jump if A ≠ 0
JZ	Jump if A = 0

All conditional jumps are short jumps.

Bit level jump instructions:

Bit level JUMP instructions will check the conditions of the bit and if condition is true, it jumps to the address specified in the instruction. All the bit jumps are relative jumps.

```
JB bit, rel     ; jump if the direct bit is set to the relative address specified.
JNB bit, rel    ; jump if the direct bit is clear to the relative address specified.
JBC bit, rel    ; jump if the direct bit is set to the relative address specified and then clear the bit.
```

BIT MANIPULATION INSTRUCTIONS.

8051 has 128 bit addressable memory. Bit addressable SFRs and bit addressable PORT pins. It is possible to perform following bit wise operations for these bit addressable locations.

1. LOGICAL AND
   a. ANL C,BIT(BIT ADDRESS) ; 'LOGICALLY AND' CARRY AND CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
   b. ANL C, /BIT; ; 'LOGICALLY AND' CARRY AND COMPLEMENT OF CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY

2. LOGICAL OR
   a. ORL C,BIT(BIT ADDRESS) ; 'LOGICALLY OR' CARRY AND CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
   b. ORL C, /BIT; ; 'LOGICALLY OR' CARRY AND COMPLEMENT OF CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
3. CLR bit
   a. CLR bit ; CONTENT OF BIT ADDRESS SPECIFIED WILL BE CLEARED.
   b. CLR C ; CONTENT OF CARRY WILL BE CLEARED.
4. CPL bit
   a. CPL bit ; CONTENT OF BIT ADDRESS SPECIFIED WILL BE COMPLEMENTED.
   b. CPL C ; CONTENT OF CARRY WILL BE COMPLEMENTED.

2.4 SIMPLE ASSEMBLY LEVEL PROGRAMS USING ABOVE INSTRUCTIONS

1. ***Write a program to add the values of locations 50H and 51H and store the result in locations in 52h and 53H.***

```
ORG 0000H          ; Set program counter 0000H
MOV A,50H          ; Load the contents of Memory location 50H into A ADD ADD A,51H
                   ; Add the contents of memory 51H with CONTENTS A
MOV 52H,A          ; Save the LS byte of the result in 52H
MOV A, #00         ; Load 00H into A
ADDC A, #00        ; Add the immediate data and carry to A
MOV 53H,A          ; Save the MS byte of the result in location 53h
END
```

2. ***Write a program to store data FFH into RAM memory locations 50H to 58H using direct addressing mode***

```
ORG 0000H          ; Set program counter 0000H
MOV A, #0FFH       ; Load FFH into A
MOV 50H, A         ; Store contents of A in location 50H
MOV 51H, A         ; Store contents of A in location 5IH
MOV 52H, A         ; Store contents of A in location 52H
MOV 53H, A         ; Store contents of A in location 53H
MOV 54H, A         ; Store contents of A in location 54H
MOV 55H, A         ; Store contents of A in location 55H
MOV 56H, A         ; Store contents of A in location 56H
MOV 57H, A         ; Store contents of A in location 57H
MOV 58H, A         ; Store contents of A in location 58H
END
```

3. ***Write a program to subtract a 16 bit number stored at locations 51H-52H from 55H-56H and store the result in locations 40H and 41H. Assume that the least significant byte of data or the result is stored in low address. If the result is positive, then store 00H, else store 01H in 42H.***

```
ORG 0000H      ; Set program counter 0000H
MOV A, 55H     ; Load the contents of memory location 55 into A
CLR C          ; Clear the borrow flag
SUBB A,51H     ; Sub the contents of memory 51H from contents of A
MOV 40H, A     ; Save the LSByte of the result in location 40H
MOV A, 56H     ; Load the contents of memory location 56H into A
SUBB A, 52H    ; Subtract the content of memory 52H from the content A
MOV 41H,       ; Save the MSbyte of the result in location 415.
MOV A, #00     ; Load 005 into A
ADDC A, #00    ; Add the immediate data and the carry flag to A
MOV 42H, A     ; If result is positive, store00H, else store 0lH in 42H
END
```

4. ***Write a program to add two 16 bit numbers stored at locations 51H-52H and 55H-56H and store the result in locations 40H, 41H and 42H. Assume that the least significant byte of data and the result is stored in low address and the most significant byte of data or the result is stored in high address.***

```
ORG 0000H          ; Set program counter 0000H
MOV A,51H          ; Load the contents of memory location 51H into A
ADD A,55H          ; Add the contents of 55H with contents of A
MOV 40H,A          ; Save the LS byte of the result in location 40H
MOV A,52H          ; Load the contents of 52H into A
ADDC A,56H         ; Add the contents of 56H and CY flag with A
MOV 41H,A          ; Save the second byte of the result in 41H
MOV A,#00          ; Load 00H into A
ADDC A,#00         ; Add the immediate data 00H and CY to A
MOV 42H,A          ; Save the MS byte of the result in location 42H
END
```

5. ***Write a program to store data FFH into RAM memory locations 50H to 58H using indirect addressing mode.***

```
        ORG 0000H          ; Set program counter 0000H
        MOV A, #0FFH       ; Load FFH into A
        MOV RO, #50H       ; Load pointer, R0-50H
        MOV R5, #08H       ; Load counter, R5-08H
Start:  MOV @RO, A         ; Copy contents of A to RAM pointed by R0
        INC RO             ; Increment pointer
        DJNZ R5, start     ; Repeat until R5 is zero
        END
```

6. ***Write a program to add two Binary Coded Decimal (BCD) numbers stored at locations 60H and 61H and store the result in BCD at memory locations 52H and 53H. Assume that the least significant byte of the result is stored in low address.***

```
ORG 0000H      ; Set program counter 00004
MOV A,60H      ; Load the contents of memory location 6.0.H into A
ADD A,61H      ; Add the contents of memory location 61H with contents of A
DA A           ; Decimal adjustment of the sum in A
MOV 52H, A     ; Save the least significant byte of the result in location 52H
MOV A,#00      ; Load 00H into .A
ADDC A,#00H    ; Add the immediate data and the contents of carry flag to A
MOV 53H,A      ; Save the most significant byte of the result in location 53:,
END
```

7. ***Write a program to clear 10 RAM locations starting at RAM address 1000H.***

```
        ORG 0000H           ;Set program counter 0000H
        MOV DPTR, #1000H    ;Copy address 1000H to DPTR
        CLR A               ;Clear A
        MOV R6, #0AH        ;Load 0AH to R6
again:  MOVX @DPTR,A        ;Clear RAM location pointed by DPTR
        INC DPTR            ;Increment DPTR
        DJNZ R6, again      ;Loop until counter R6=0
        END
```

8. ***Write a program to compute 1 + 2 + 3 + N (say N=15) and save the sum at70H***

```
        ORG 0000H               ; Set program counter 0000H
        N EQU 15
        MOV R0,#00              ; Clear R0
        CLR A                   ; Clear A
again:  INC R0                  ; Increment R0
        ADD A, R0               ; Add the contents of R0 with A
        CJNE R0,#N,again        ; Loop until counter, R0, N
        MOV 70H,A               ; Save the result in location 70H END
```

9. ***Write a program to multiply two 8 bit numbers stored at locations 70H and 71H and store the result at memory locations 52H and 53H. Assume that the least significant byte of the result is stored in low address.***

ORG 0000H ; Set program counter 00 OH
MOV A, 70H ; Load the contents of memory location 70h into A
MOV B, 71H ; Load the contents of memory location 71H into B
MUL AB ; Perform multiplication
MOV 52H,A ; Save the least significant byte of the result in location 52H MOV 53H,B ; Save the most significant byte of the result in location 53
END

10. ***Ten 8 bit numbers are stored in internal data memory from location 5oH. Write a program to increment the data.***
Assume that ten 8 bit numbers are stored in internal data memory from location 50H, hence R0 or R1 must be used as a pointer.

The program is as follows.

```
        OPT 0000H
        MOV R0,#50H
        MOV R3,#0AH
        Loopl: INC @R0
        INC RO
        DJNZ R3, loopl END
        END
```

11. Write a program to find the average of five 8 bit numbers. Store the result in H. (Assume that after adding five 8 bit numbers, the result is 8 bit only).

```
ORG 0000H
MOV 40H,#05H
MOV 41H,#55H
MOV 42H,#06H
MOV 43H,#1AH
MOV 44H,#09H
MOV R0,#40H
MOV R5,#05H
MOV B,R5
CLR A
Loop: ADD A,@RO
INC RO
DJNZ R5,Loop
DIV AB
MOV 55H,A                    END
```

12. Write a program to find the cube of an 8 bit number program is as follows

```
ORG 0000H
MOV R1,#N
MOV A,R1
MOV B,R1
MUL AB                  //SQUARE IS COMPUTED
MOV R2, B
MOV B, R1
MUL AB
MOV 50,A
MOV 51,B
MOV A,R2
MOV B, R1
MUL AB
ADD A, 51H
MOV 51H, A
MOV 52H, B
MOV A, # 00H
ADDC A, 52H
MOV 52H, A              //CUBE IS STORED IN 52H,51H,50H
END
```

13. Write a program to exchange the lower nibble of data present in external memory 6000H and 6001H

```
ORG 0000H           ; Set program counter 00h
MOV DPTR, #6000H    ; Copy address 6000H to DPTR
MOVX A, @DPTR       ; Copy contents of 60008 to A
MOV R0, #45H        ; Load pointer, R0=45H
MOV @RO, A          ; Copy cont of A to RAM pointed by 80
INC DPL             ; Increment pointer
MOVX A, @DPTR       ; Copy contents of 60018 to A
```

```
XCHD A, @R0          ; Exchange lower nibble of A with RAM pointed by RO
MOVX @DPTR, A        ; Copy contents of A to 60018
DEC DPL              ; Decrement pointer
MOV A, @R0           ; Copy cont of  RAM pointed by R0 to A
MOVX @DPTR, A        ; Copy cont of A to RAM pointed by DPTR
END
```

14. Write a program to count the number of and o's of 8 bit data stored in location 6000H.

```
ORG 00008                    ; Set program counter 00008
MOV DPTR, #6000h      ; Copy address 6000H to DPTR
MOVX A, @DPTR                ; Copy number to A
MOV R0,#08                   ; Copy 08 in RO
MOV R2,#00                   ; Copy 00 in R2
MOV R3,#00                   ; Copy 00 in R3
CLR C                        ; Clear carry flag
BACK:  RLC A          ; Rotate A through carry flag
JC NEXT               ; If CF = 1, branch to next
INC R2                ; If CF = 0, increment R2 AJMP NEXT2
NEXT:  INC R3         ; If CF = 1, increment R3
NEXT2: DJNZ RO,BACK ; Repeat until RO is zero
END
```

15. Write a program to shift a 24 bit number stored at 57H-55H to the left logically four places. Assume that the least significant byte of data is stored in lower address.

```
        ORG 0000H      ; Set program counter 0000h
        MOV R1,#04     ; Set up loop count to 4
again:  MOV A,55H      ; Place the least significant byte of data in A
        CLR C          ; Clear tne carry flag
        RLC A          ; Rotate contents of A (55h) left through carry
        MOV 55H,A
        MOV A,56H
        RLC A          ; Rotate contents of A (56H) left through carry
        MOV 56H,A
        MOV A,57H
        RLC A          ; Rotate contents of A      (57H) left through carry
        MOV 57H,A
        DJNZ R1,again  ; Repeat until R1 is zero
        END
```