VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY AND RESEARCH

(Deemed to-be University U/s 3 of the UGC Act, 1956)

Micro Processors & Interfacing

16CS307

Unit- 4

Lec-3

Mr. M Krishna Chennakesava Rao,

Asst. Professor, Dept. of ECE,

VFSTR University

Instruction Set of 8051

- All instructions of 8051 can be divided in to four different groups.

1. **Data Transfer Instructions**
2. **Arithmetic Instructions**
3. **Logical Instructions**
4. **Branching Instructions**

Instruction Set of 8051 cont'd…

1. Data Transfer Instructions:

- Also called as Data Movement or Data Copying Instructions.
- Data transfer instruction execution **doesn't effect** on the **Mathematical flags** in PSW.
- Data transfer instruction are **three** types.

A. Move Related Instructions : MOV, MOVX, MOVC

B. Stack Related Instructions : PUSH, POP

C. Exchange Related Instructions : XCH, XCHD

Instruction Set of 8051 cont'd…

1. Data Transfer Instructions:

A. Move Related Instructions: -

- There are **three** types move related instructions.

  i. **MOV** ii. **MOVX** iii. **MOVC**

i. MOV Instruction:-

This instruction is used for **onboard data transfers** in 8051 μC.

Ex: -

```
MOV R1, #68H
MOV A, #88H
MOV A, R0
MOV DPTR, #0896H
```

Instruction Set of 8051 cont'd…

1. Data Transfer Instructions:

A. Move Related Instructions: -

ii. MOVX Instruction: -

✓ It is used for data transfers b/w **external RAM** and **µC** through 'A' register.

✓ This instruction only supports **Register Indirect Addressing Mode.**

Ex: - MOVX A, @R_P

MOVX A, @DPTR

MOVX @R_P, A

MOVX @DPTR, A

iii. MOVC Instruction: - Move Code Byte

✓ It is used for data transfers b/w **internal or external ROM** and **µC** through 'A' register.

✓ This instruction only supports **Register Indexed Addressing Mode.**

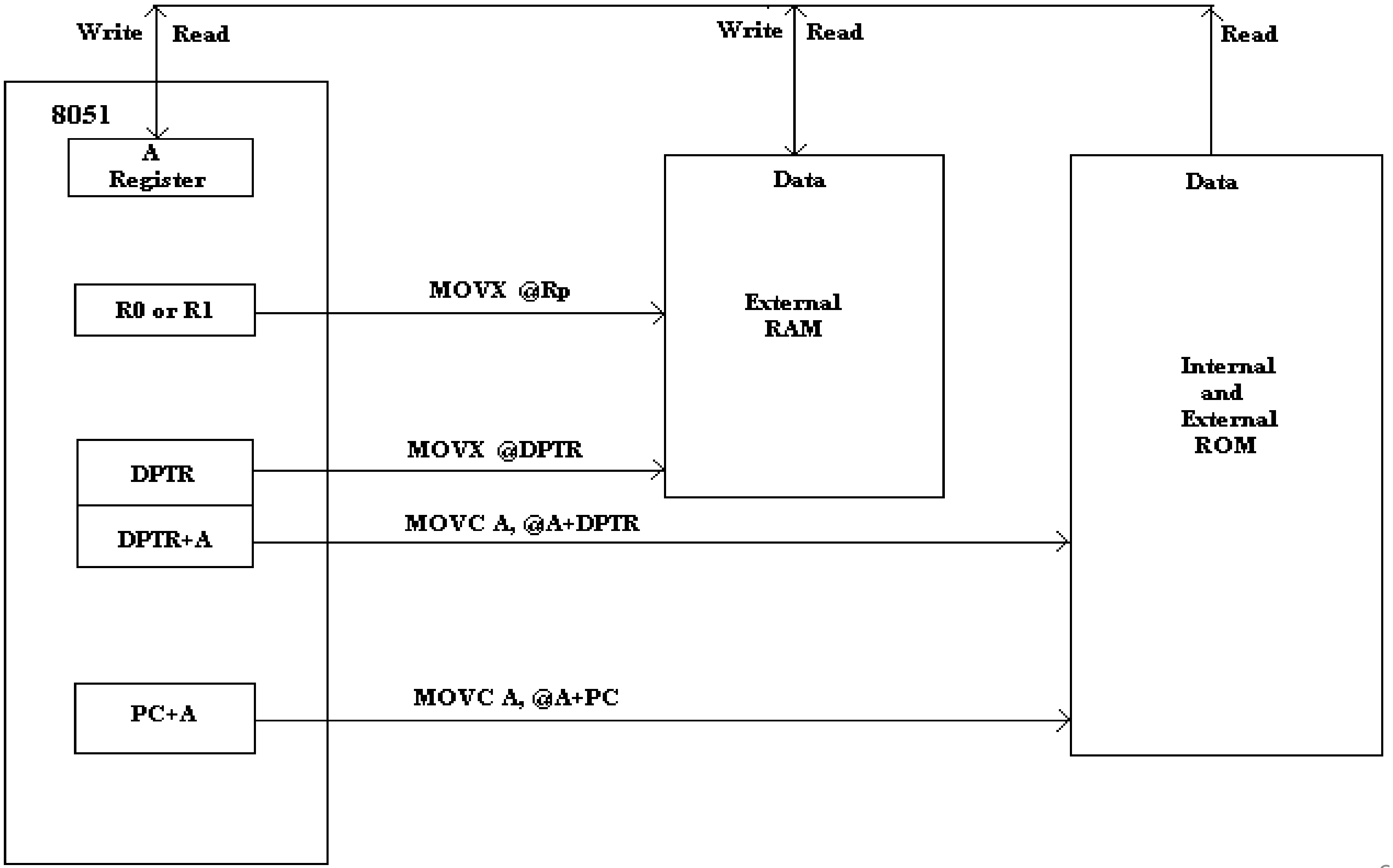
- Ex: - MOVC A, @A+DPTR

  MOVC A, @A+PC

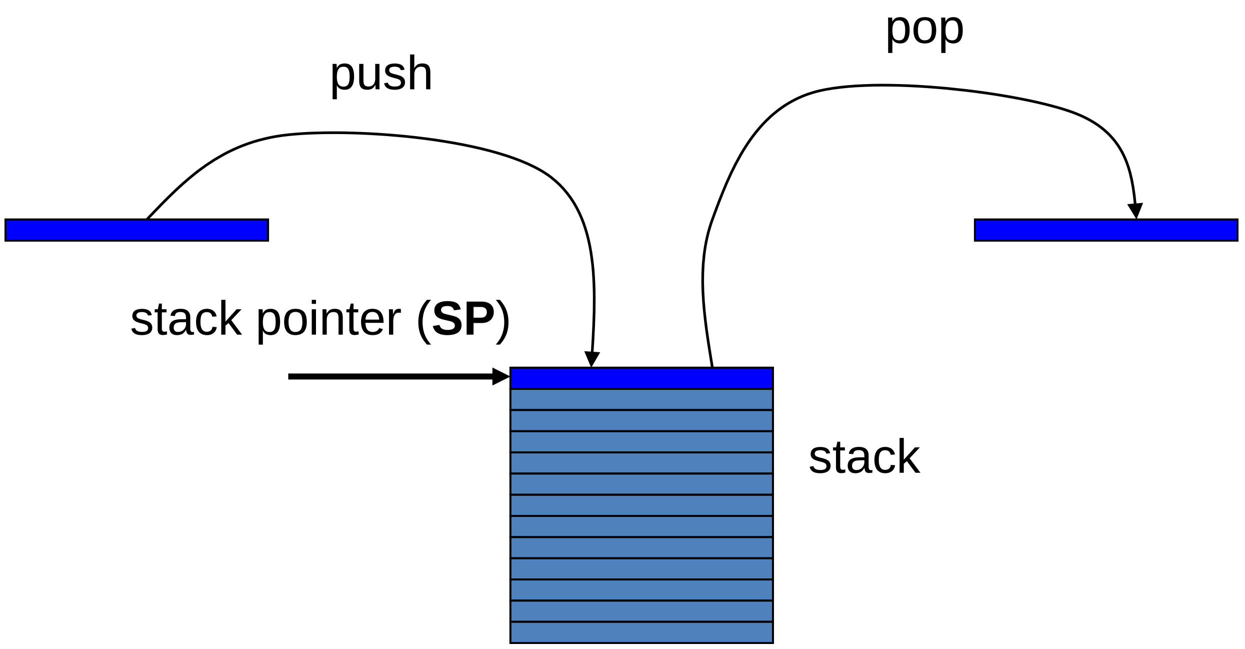
Instruction Set of 8051 cont'd…

1. Data Transfer Instructions:

A. Move Related Instructions: -



Stacks



Go do the stack exercise.....

Instruction Set of 8051 cont'd…

1. Data Transfer Instructions:

B. Stack Related Instructions: -

i. PUSH Instruction: -

- ✓ It is used for transferring the data from specified **Direct Address into Top of the Stack.**
- ✓ For this instruction execution the **SP content is increment by '1'.**
- ✓ It support only **Direct Addressing Mode**
- ✓ In this instruction execution the internal operations are performed in the following order.

    1. SP content increment by '1'.
    2. Specified Direct Address content is pushed on to SP specified location in the Stack.

- Ex: - **PUSH 76H**

Instruction Set of 8051 cont'd…

1. Data Transfer Instructions:

B. Stack Related Instructions: -

ii. POP Instruction: -

- ✓ It is used for transferring the data from Top of the Stack to specified Direct Address.
- ✓ For this instruction execution the SP content is decrement by '1'.
- ✓ It supports only direct Addressing Mode.
- ✓ In this instruction execution the internal operations are performed in the following order.

1. SP specified location in the Stack content is copied to Specified Direct Address in instruction.

2. SP content decrement by '1'

- Ex: - **POP 86H**

Instruction Set of 8051 cont'd…

1. Data Transfer Instructions:

C. Exchange Related Instructions:

- In these instructions one operand is accumulator register 'A'.
- These instructions **don't support Immediate Addressing Mode**.

i. **XCH** ii. **XCHD**

i. XCH Instruction: -

✓ It is used for data exchanging b/w Accumulator register 'A' and the specified other operand in the instruction.

✓ Ex: -

```
XCH   A, R0
XCH   A, @R1
XCH   A, 46H
```

Instruction Set of 8051 cont'd…

1. Data Transfer Instructions:

C. Exchange Related Instructions:

ii. XCHD Instruction: -

- ✓ It is used for **Least Significant Nibbles data exchanging** between Accumulator register 'A' and the specified other operand in the instruction.

- Ex: -

```
XCHD   A, R0
XCHD   A, @R1
XCHD   A, 46H
```

Instruction Set of 8051 cont'd…

2. Arithmetic Instructions :

- ✓ If any arithmetic instruction is executed by controller, based on the result the **mathematical flags are modified**.
- ✓ For these arithmetic instructions one operand must be the content of accumulator 'A' .
- ✓ After completion of operation the **result is stored in register 'A'**.

i. Addition **ii. Subtraction** **iii. Multiplication**

iv. Division **v. Increment** **vi. Decrement**

vii. Decimal Adjust

Instruction Set of 8051 cont'd...

2. Arithmetic Instructions :

i. Addition Instructions: -

- There are two instructions for to perform addition operation.

  a. ADD b. ADDC

a. ADD Instruction: -

- 8-bit addition between the accumulator (A) and a second operand.
  - The result is always in the accumulator.
  - The CY flag is set/reset appropriately.
  - It supports all addressing modes.

✓ Ex: - **ADD A, R0** **ADD A, #24H** **ADD A, @R0** **ADD A, 68H**

Addition Instructions: -

b. ADDC Instruction: -

- 8-bit addition between the accumulator, a second operand and the **previous value of the CY flag.**
  - Useful for 16-bit addition in two steps.
  - The CY flag is set/reset appropriately.
  - It supports all addressing modes.

Ex: -

```
ADDC   A, R0        ADDC   A, #24H
ADDC   A, @R0       ADDC   A, 68H
```

Example – 16-bit Addition

Add 1E44H to 56CAH

```
CLR    C            ; Clear the CY flag
MOV    A, 44H       ; The lower 8-bits of the 1st number
ADD    A, CAH       ; The lower 8-bits of the 2nd number
MOV    R1, A        ; The result 0EH will be in R1. CY = 1.
MOV    A, 1EH       ; The upper 8-bits of the 1st number
ADDC   A, 56H       ; The upper 8-bits of the 2nd number
MOV    R2, A        ; The result of the addition is 75H
```

The overall result: 750EH will be in R2:R1. CY = 0.

ADD Instructions

```
add a, byte          ; a ← a + byte
addc a, byte              ; a ← a + byte + C
```

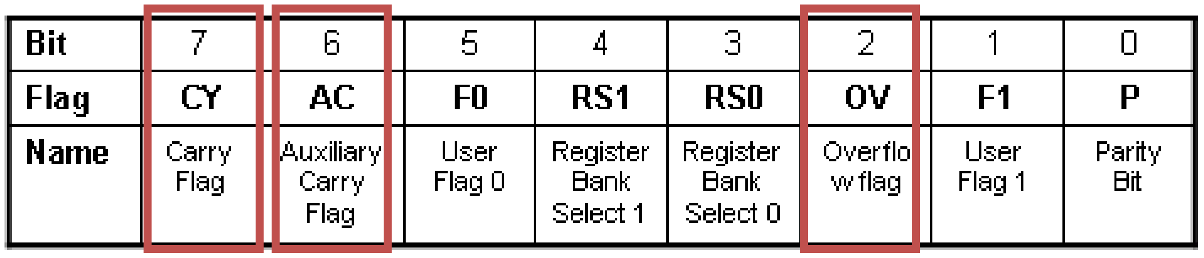
These instructions affect 3 bits in PSW:

C = 1 if result of add is greater than FF

AC = 1 if there is a carry out of bit 3

OV = 1 if there is a carry out of bit 7, but not from bit 6, or visa versa.

Program Status Word (PSW)



Bit	7	6	5	4	3	2	1	0
Flag	**CY**	**AC**	**F0**	**RS1**	**RS0**	**OV**	**F1**	**P**
Name	Carry Flag	Auxiliary Carry Flag	User Flag 0	Register Bank Select 1	Register Bank Select 0	Overflo w flag	User Flag 1	Parity Bit

Instruction Set of 8051 cont'd…

2. Arithmetic Instructions :

ii. Subtraction Instruction: -

- ✓ Only one instruction is there to perform the subtraction operation.
- ✓ That is subtraction with borrow '**SUBB**'.
- ✓ For this **Accumulator** content is **Minuend**.

- **SUBB Instruction: -**

- ✓ It is used to perform subtraction between Accumulator & specified another operand in the instruction and previous operation generated carry.
- ✓ It supports all addressing modes.

- ✓ Ex: -

```
SUBB   A, R0          SUBB   A, #24H
SUBB   A, @R0         SUBB   A, 68H
```

Instruction Set of 8051 cont'd…

2. Arithmetic Instructions :

iii. MUL Instruction: -

- ✓ It is used to perform Multiplication between 'A' register and 'B' register.
- ✓ The result of the Multiplication is stored in B & A registers.
- ✓ MSByte is stored in 'B' register and LSByte is stored in 'A' register.
- ✓ It support only two 8-bits numbers Multiplication

Ex: - **MUL AB**

iv. DIV Instruction: --

- ✓ It is used to perform Division between 'A' register and 'B' register.
- ✓ The result of the Division is stored in A & B registers.
- ✓ Quotient is stored in 'A' register and Remainder is stored in 'B' register.

Ex: - **DIV AB**

Instruction Set of 8051 cont'd…

2. Arithmetic Instructions :

v. INC (Increment) Instruction: -

- ✓ It supports all types of addressing modes except Immediate AM.
- ✓ No mathematical flag is affected for this instruction execution.
- ✓ It is used to increment the specified operand content by '1'.
- ✓ The operand may not an immediate data.

Ex: - **INC R0** **INC 70H** **INC @R0** **INC DPTR** **INC A**

Instruction Set of 8051 cont'd…

2. Arithmetic Instructions :

vi. DEC (Decrement) Instruction: -

- ✓ It supports all types of addressing modes except Immediate AM.
- ✓ No mathematical flag is affected for this instruction execution.
- ✓ It is used to decrement the specified operand content by '1'.
- ✓ The operand may not an immediate data.

Ex: - **DEC R0** **DEC 70H** **DEC @R0** **DEC DPTR** **DEC A**

Increment and Decrement

`INC A`	`increment A`
`INC byte`	`increment byte in memory`
`INC DPTR`	`increment data pointer`
`DEC A`	`decrement accumulator`
`DEC byte`	`decrement byte`

- The increment and decrement instructions do NOT affect the C flag.
- Notice we can only INCREMENT the data pointer, not decrement.

Instruction Set of 8051 cont'd…

2. Arithmetic Instructions :

vii. Decimal Adjust Instruction: -

- ✓ It supports only register addressing mode.
- ✓ Operates only on the accumulator.
- ✓ Works only after the ADD instruction.

- **DA A Instruction: -**
- ✓ It is used to decimal adjust the accumulator content, it means **convert the binary content of register 'A' into BCD form**.
-
- Ex: - **DA A**

Example – BCD addition

Add **34** to **49** **BCD**

```
CLR   C            ; Clear the CY flag
MOV   A, #34H      ; Place 1st number in A
ADD   A, #49H      ; Add the 2nd number.
                   ; A = 7DH
DA    A            ; A = 83H
```

Instruction Set of 8051 cont'd…

2. Arithmetic Instructions :

Mnemonic	Description
`ADD A, byte`	add A to byte, put result in A
`ADDC A, byte`	add with carry
`SUBB A, byte`	subtract with borrow
`INC A`	increment A
`INC byte`	increment byte in memory
`INC DPTR`	increment data pointer
`DEC A`	decrement accumulator
`DEC byte`	decrement byte
`MUL AB`	multiply accumulator by b register
`DIV AB`	divide accumulator by b register
`DA A`	decimal adjust the accumulator

Instruction Set of 8051 cont'd…

3. Logical Instructions :

- **Byte level** Logical Instructions,
- **Bit Level** Logical Instructions,

Instruction Set of 8051 cont'd…

3. Logical Instructions :

1. **Byte level** logical operations.

i. **ANL** (Logical AND)	ii. **ORL** (Logical OR)
iii. **XRL** (Logical XOR)	iv. **CPL** (Logical NOT)

Rotation related Instructions.

i. Rotation Left (**RL**)
ii. Rotation Right (**RR**)
iii. Rotation Left by Carry (**RLC**)
iv. Rotation Right by Carry (**RRC**)
v. **Swap** Instruction

Instruction Set of 8051 cont'd…

3. Logical Instructions :

2. Bit level logical operations.

i. AND Related Instructions :	a. ANL C, b	b. ANL C, b*
ii. OR Related Instructions :	a. ORL C, b	b. ORL C, b*
iii. Complement Related Instructions :	a. CPL C	b. CPL b
iv. Clear Related Instructions :	a. CLR C	b. CLR b
v. Set Related Instructions:	a. SETB C	b. SETB b
vi. Move Related Instructions:	a. MOV C,b	b. MOV b, C

Instruction Set of 8051 cont'd…

3. Logical Instructions : (Byte Level)

- **ANL / ORL / XRL**
  - Work on byte sized operands .
  - It supports all addressing modes.

```
ANL  A, R0
ANL  70H, R1
ANL  A, #80H
ANL  A, @R0
```

```
ORL   A, R0
ORL   70H, R1
ORL   A, #80H
ORL   A, @R0
```

```
XRL   A, R0
XRL   70H, R1
XRL   A, #80H
XRL   A, @R0
```

- **CPL Instruction: -** Performs logical NOT operation

Ex: **CPL A**

Instruction Set of 8051 cont'd…

3. Logical Instructions : (Byte Level)

- **RL / RLC / RR / RRC**
  - Rotate the accumulator.
    - RL and RR without the carry
    - RLC and RRC rotate through the carry.

- **SWAP A**
  - Swap the upper and lower nibbles of the accumulator.

Instruction Set of 8051 cont'd…

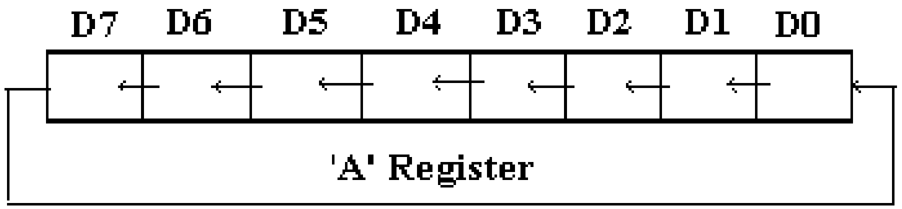
3. Logical Instructions : (Byte Level)

RL Instruction:

✓ used to rotate the accumulator content bit by bit to left side.

✓ MSB is copied into LSB.

Ex: - RL A

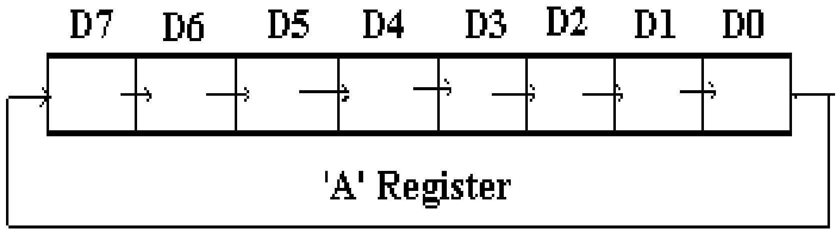


RR Instruction:

✓ used to rotate the accumulator content bit by bit to right side.

✓LSB is copied into MSB.

Ex: - RR A



Instruction Set of 8051 cont'd…

3. Logical Instructions : (Byte Level)

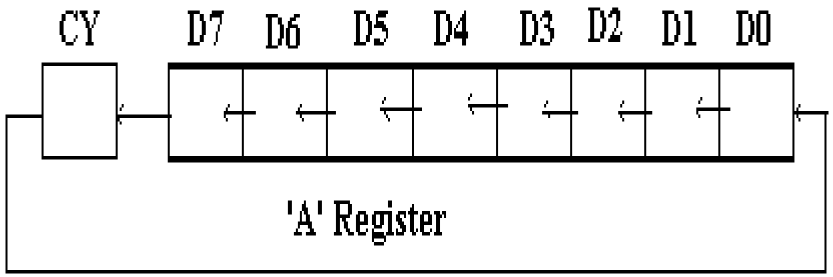
- **RLC Instruction: -**

✓ used to rotate the accumulator content bit by bit to left side through CY.

✓ MSB is copied to Carry flag position and

✓ Carry bit is copied to LSB position.

- Ex: - RLC A



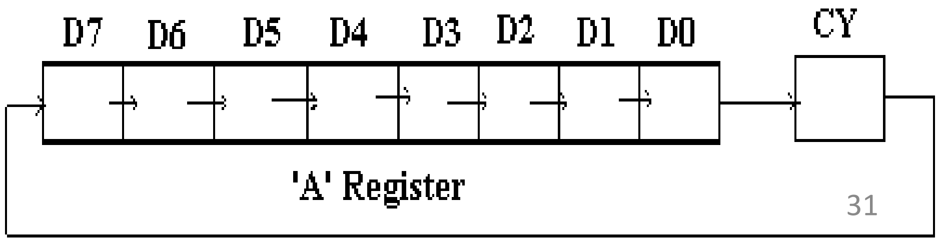
- **RRC Instruction: -**

✓ used to rotate the accumulator content bit by bit to Right side through CY.

✓ LSB is copied to Carry flag position and

✓ Carry bit is copied to MSB position.

- Ex: - RRC A



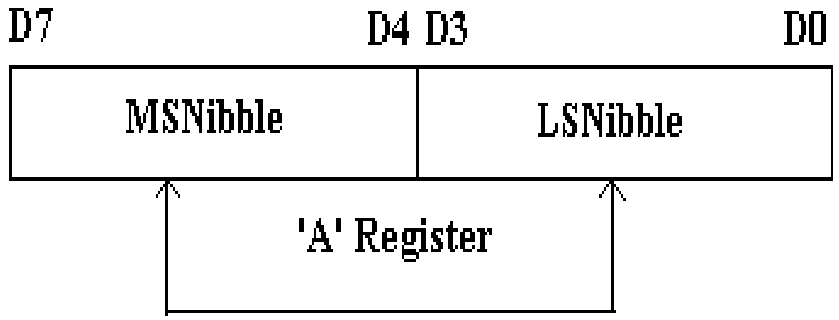
Instruction Set of 8051 cont'd...

3. Logical Instructions : (Byte Level)

SWAP Instruction: -

✓ used to exchange the nibbles of Accumulator register.

Ex: - **SWAP a**



```
mov a, #72h        ; a ← 72h
swap a             ; a ← 27h
```

Instruction Set of 8051 cont'd…

3. Logical Instructions : (Bit Level)

- This group of instructions is associated with the single-bit operations of the 8051.
- This group allows manipulating the individual bits of bit addressable registers and memory locations as well as the CY flag.
  - The P, OV, and AC flags cannot be directly altered.

- **This group includes:**
  - Set, clear, and, or, complement , move.
  - Conditional jumps.

Instruction Set of 8051 cont'd…

3. Logical Instructions : (Bit Level)

2. Bit level logical operations.

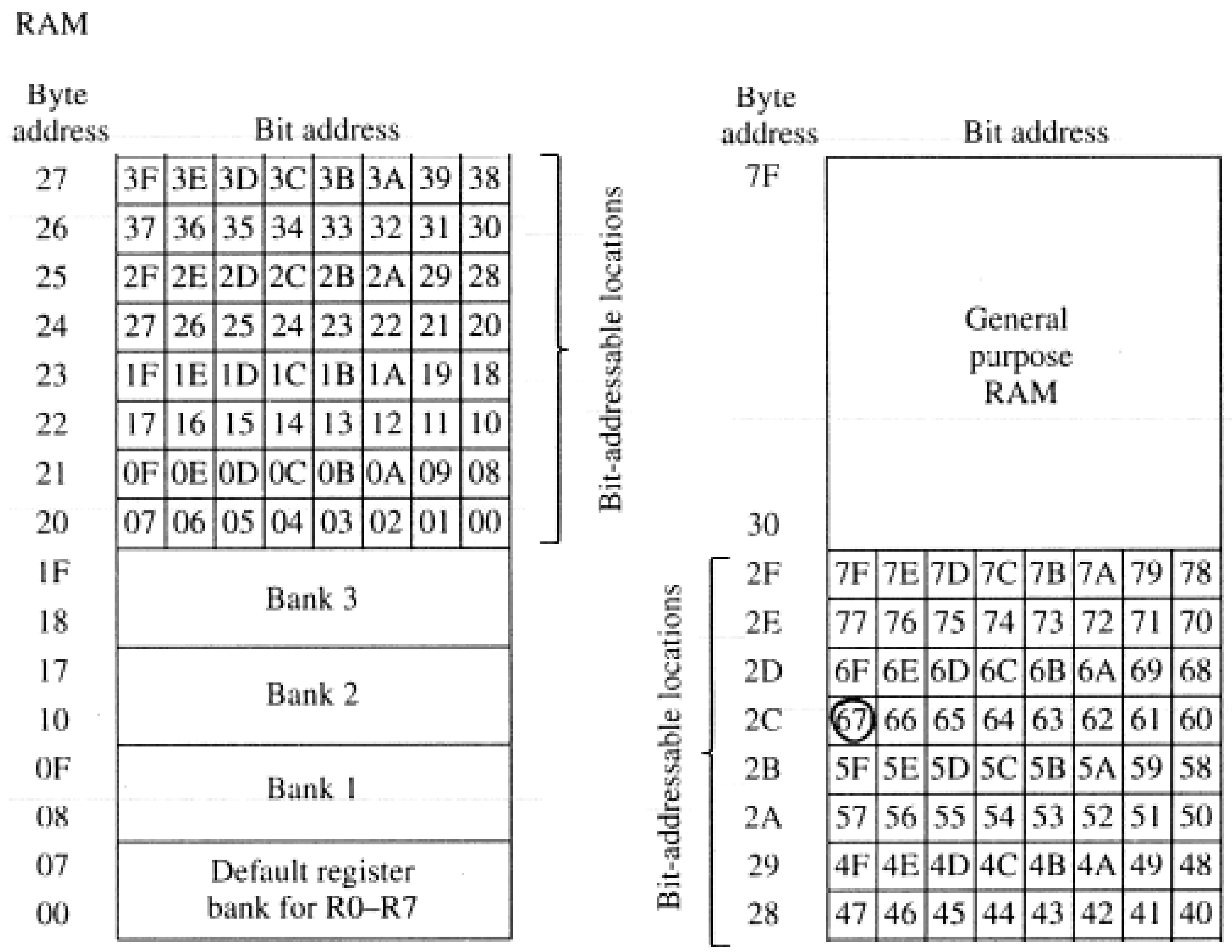
i. AND Related Instructions :	a. ANL C, b	b. ANL C, b*
ii. OR Related Instructions :	a. ORL C, b	b. ORL C, b*
iii. Complement Related Instructions :	a. CPL C	b. CPL b
iv. Clear Related Instructions :	a. CLR C	b. CLR b
v. Set Related Instructions:	a. SETB C	b. SETB b
vi. Move Related Instructions:	a. MOV C,b	b. MOV b, C

Bit Addressable RAM

Figure 2-6

Summary of the 8051 on-chip data memory

(RAM)



Byte address	Bit address							
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00
1F	Bank 3							
18								
17	Bank 2							
10								
0F	Bank 1							
08								
07	Default register bank for R0–R7							
00								

Byte address	Bit address							
7F	General purpose RAM							
30								
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40

Instruction Set of 8051 cont'd…

3. Logical Instructions : (Bit Level)

AND / OR Related Instructions

❖ **OR / AND a bit with the CY flag.**

- **i. ANL C, b   ORL C, b**

✓ logical AND/ OR between Carry Flag (CY) and the specified direct addressed bit in the instruction.

✓ Ex: - **ANL C, 67H.**   **ORL C, 67H**

ii. ANL C, b*   **ORL C, b***

✓ logical AND between Carry Flag (CY) and the complement of specified direct addressed bit.

✓ Ex: - **ANL C, 67*H**   **ORL C, 67*H**

Instruction Set of 8051 cont'd…

3. Logical Instructions : (Bit Level)

- ORL / ANL
  - OR / AND a bit with the CY flag.
    - ORL C, 20H ; OR bit 20 of bit addressable memory with the CY flag
    - ANL C, 34*H ; AND complement of bit 34 of bit addressable memory with the CY flag.

Instruction Set of 8051 cont'd…

3. Logical Instructions : (Bit Level)

- **MOV**
  - **Data transfer between a bit and the CY flag.**
    - **MOV C, 3FH** ; Copy the bit 3F of the bit addressable memory to CY flag
    - **MOV P1.2, C** ; Copy the CY flag to bit 2 of P1.

Instruction Set of 8051 cont'd…

3. Logical Instructions : (Bit Level)

- **CLR**
  - **Clear a bit or the CY flag.**
    - CLR P1.1
    - CLR C

- **SETB**
  - **Set a bit or the CY flag.**
    - SETB P1.2
    - SETB C

- **CPL**
  - **Complement a bit or the CY flag.**
    - CPL 40H ; Complement bit 40 of the bit addressable memory

3. Bit Logic Operations

- Some logic operations can be used with single bit operands

```
ANL C, bit
ORL C, bit
CLR C
CLR bit
CPL C
CPL bit
SETB C
SETB bit
Mov C, bit
```

- “bit” can be any of the bit-addressable RAM locations or SFRs.

4. Program Flow Control Or Branching Instructions :

- By using these instructions the **program flow control** **is** **transferred from one location to another location** conditionally or unconditionally.

- **3 types**

1. **Unconditional jumps ("go to")** **: SJMP, LJMP, AJMP, JMP**
2. **Conditional jumps**
3. **Call and return**

4. Branching Instructions :Unconditional Jumps

- `SJMP <relative addr>   ; Short jump,`

✓ relative address is 8-bit 2's complement number,

✓ so jump can be up to 127 locations forward, or 128 locations back.

- `LJMP <address 16> ; Long jump`

✓ To anywhere within **64K** block/locations of program memory i.e.,

✓ The destination must be within **64K** locations from the current instruction.

✓ **Execution is moved to only next locations.**

✓ The operand is an **16-bit address.**

4. Branching Instructions :Unconditional Jumps

- **AJMP <address 11>** ; **Absolute jump**

✓ To anywhere within 2K block/locations of program memory i.e.,

✓ The destination must be within 2K locations from the current instruction.

✓ **Execution is moved to only next locations.**

✓ The operand is an **11-bit address**

- **JMP @A + DPTR** ; **Long indexed jump**

✓ By using this jump instruction execution is moved to the location address is provided by the combination of 'A' register and DPTR register.

4. Branching Instructions :Conditional Jumps

- These instructions cause a jump to occur only if a condition is true. Otherwise, program execution continues with the next instruction.

```
 loop: mov A, P1
       jz  loop       ; if a=0, goto loop,
                      ; else goto next instruction
       mov B, A
```

- There is no zero flag (z)
- Content of A checked for zero on time

4. Branching Instructions: Conditional Jumps cont'd…

Mnemonic	Description
JZ <rel addr>	Jump if a = 0
JNZ <rel addr>	Jump if a != 0
JC <rel addr>	Jump if C = 1
JNC <rel addr>	Jump if C != 1
JB <bit>, <rel addr>	Jump if bit = 1
JNB <bit>,<rel addr>	Jump if bit != 1
JBC <bir>, <rel addr>	Jump if bit =1, & clear bit
CJNE A, direct, <rel addr>	Compare A and memory, jump if not equal

4. Branching Instructions: Conditional Jumps cont'd…

Mnemonic	Description
CJNE A, #data <rel addr>	Compare A and data, jump if not equal
CJNE Rn, #data <rel addr>	Compare Rn and data, jump if not equal
CJNE @Rn, #data <rel addr>	Compare data@Rn and data in memory, jump if not equal
DJNZ Rn, <rel addr>	Decrement Rn and then jump if not zero
DJNZ direct, <rel addr>	Decrement memory and then jump if not zero

Ex: CJNE A, #70H, CJNE A, 80H, CJNE Rn, #78H, CJNE @Ri, #32H

4. Branching Instructions: CALL & RETURN

- ✓ By using the **Call and Return** instructions the subprogram is called into the main program.

- ✓ **CALL** instruction is used to change the program execution, **from the main program to subprogram.**

- ✓ Call is similar to a jump, but Call **pushes PC** on stack before branching

- ✓ By using **RET** instruction the program execution is shifted, **from the subprogram to the main program**.

- ✓ There are **two CALL** related instructions,

  i. **ACALL** ii. **LCALL**

4. Branching Instructions:

CALL Instructions:

i. ACALL:

- ✓ To call a subprogram into the main program.
- ✓ But the subprogram must be within the **2K** locations.

Ex: **ACALL <11-bit address>** ; stack ← PC
; PC ← address 11 bit

ii. LCALL :

- ✓ To call a subprogram into the main program.
- ✓ The subprogram is in any location in the **64K** locations.

Ex: **LCALL <16-bit address>** ; stack ← PC
; PC ← address 16 bit

4. Branching Instructions:

Return (RET) Instruction:

- Return is also similar to a jump, but
  - Return instruction pops PC from stack
  - By using this instruction the program execution is shifted **from the subprogram to the main program**,
  - It is used as last instruction in the subprogram.
  - It comes under implicit addressing mode.
  - Ex: - **RET** ; PC ← stack
  - **RET I** ; execution shifts from ISR to Main program.
    ; It is used as last instruction in the ISR.

4. Branching Instructions: CALL & RETURN

call to the subroutine

```
 Main:      ...
            acall sublabel
            ...
            ...
sublabel:   ...
            ...
            ret
```

the subroutine

8051 - Programs

1. **Write an 8051 ALP to save the status of bits P1.1 and P1.2 on RAM bit locations 08H and 09H respectively.**

```
MOV C, P1.1
MOV 08H, C
MOV C, P1.2
MOV 09H, C
UP: SJMP UP
```

2. Write an 8051 ALP to find the sum of values 79H, 0F5H, and 0E2H. Put the sum in register R0 (Lower order Byte) and R1 (Higher order Byte).

```
MOV A, 00H
MOV R1, A
MOV R0, A
ADD A, #79H
JNC NEXT1
INC R1
NEXT1: ADD A, #0F5H
JNC NEXT 2
INC R1
NEXT 2: ADD A, #0E2H
JNC NEXT3
INC R1
NEXT 3: MOV R0, A
HERE: SJMP HERE
```

3. Write an 8051 ALP to add ‘2’ to the accumulator by 600H times.

```
MOV A, #77H
MOV R2, #10H
NEXT: MOV R3, #60H
AGAIN: ADD A, #02H
DJNZ R3, AGAIN
DJNZ R2, NEXT
STOP: SJMP STOP
```

4. Writ an 8051 ALP to toggle all the bits of Port 1 by sending to it the values 66H and 0BBH continuously. Put a delay in between each issuing of data to port.

```
UP: MOV A, #66H
MOV P1, A
LCALL DELAY
MOV A, #0BBH
MOV P1, A
LCALL DELAY
SJMP UP
DELAY: MOV R3, #0FFH
AGAIN: DJNZ R3, AGAIN
 RET
```

5. Write an 8051 ALP to find the number of 1’s in a 0FFH.

```
MOV R2, #00H
MOV R3, #08H
MOV A, #0FFH
REPEAT: RLC A
JNC NEXT
INC R2
NEXT: DJNZ R3, REPEAT
HERE: SJMP HERE
```

6. Write an 8051 ALP to find the Largest / Maximum number in the array.

```
MOV DPTR, #3000H
MOVX A, @DPTR
MOV R0, #'N'
L1: MOV B, A
L3: DJNZ R0, L2
SJMP DOWN
L2: INC DPTR
MOVX A, @DPTR
CJNE A, B, UP
SJMP L3
UP: JC L3
SJMP L1
DOWN: MOV R1, B
```