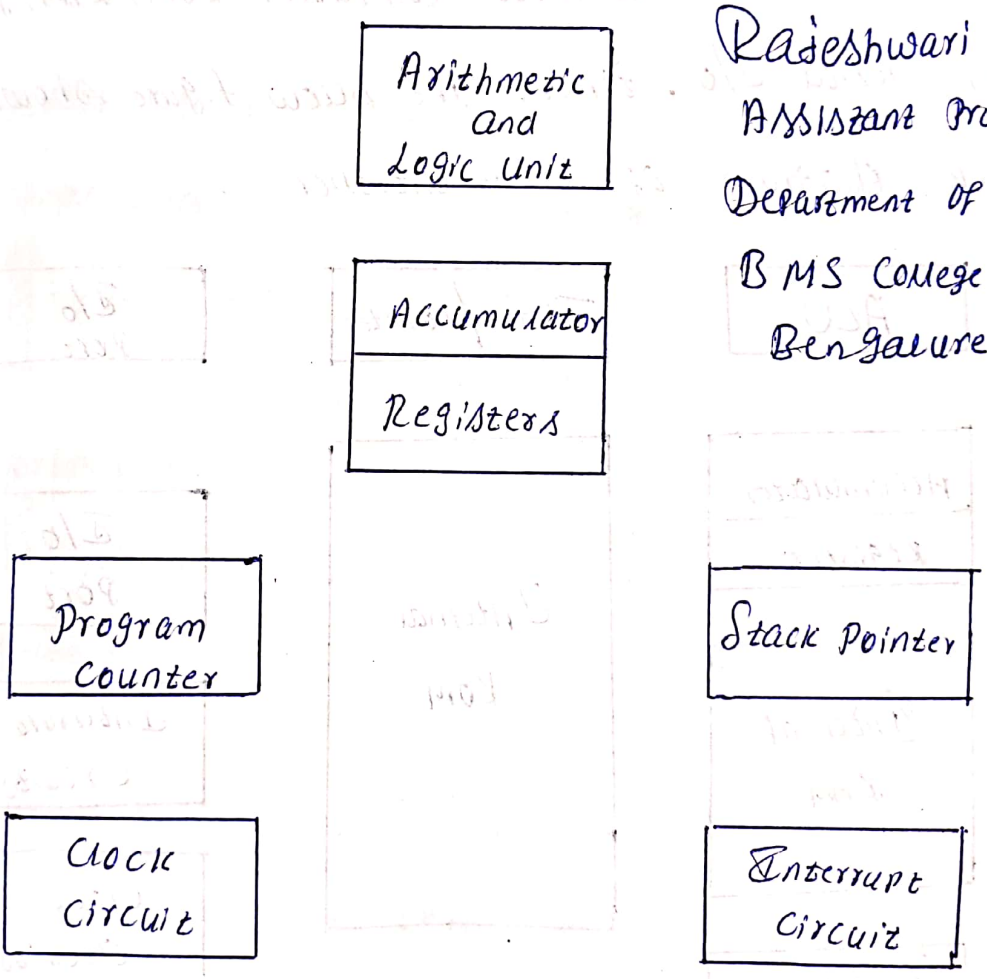
# Microprocessors and Microcontrollers

A Microprocessor is a General Purpose CPU consisting of Arithmetic and logic unit (ALU), a program counter, a stack pointer, some registers, a clock timing circuit and interrupt circuits as shown in below block diagram

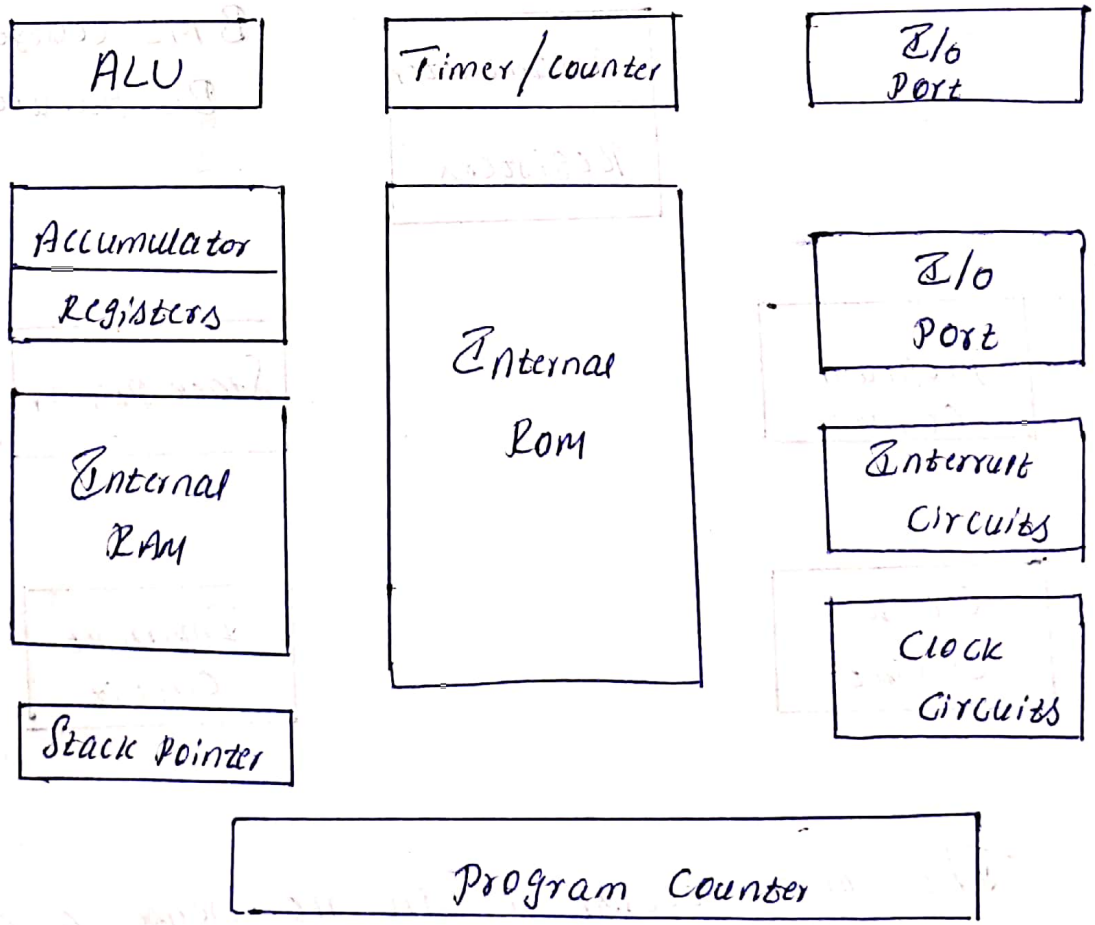
Rajeshwari BS
Assistant Professor
Department of CSE
BMS College of Engg
Bengaluru -19




The microprocessor is in no-sense a complete digital computer. To make a complete microcomputer, need to add Read Only memory (ROM), Random Access

memory (RAM), memory decoders, an Oscillator, number of input/output ports

Microcontroller is like small computer on a chip, incorporates all of the features of microprocessor like ALU, PC, SP, registers, clock circuits, interrupt circuits. It also has added the other features needed to make complete computer: ROM, RAM, Parallel I/O, Serial I/O, timer. The below figure shows block diagram of microcontroller



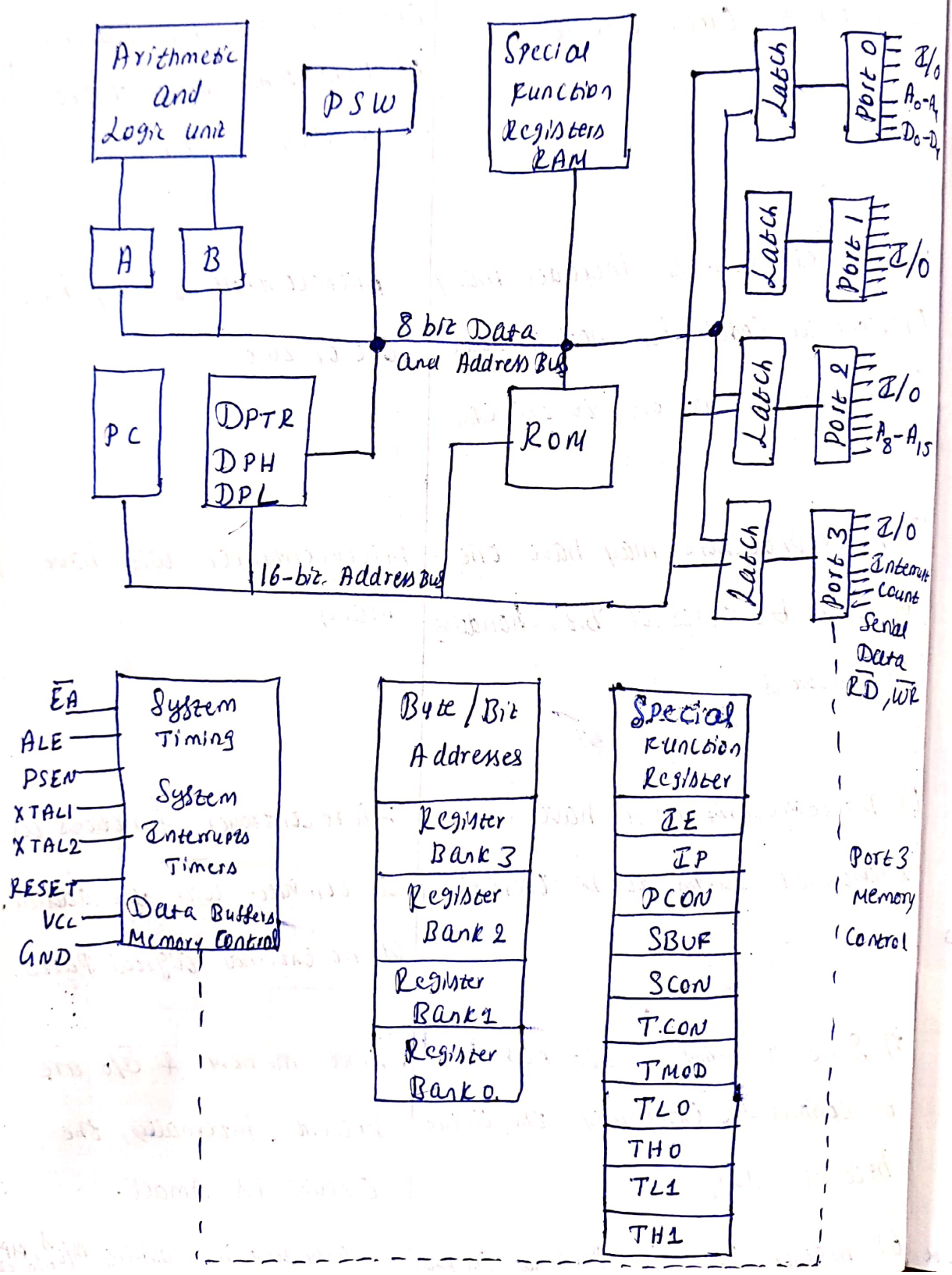
# Comparing Microprocessors and Microcontrollers

| Microprocessor | Microcontroller |
|---|---|
| 1) Microprocessor is just an integrated circuit chip consisting of ALU, CU, registers, interrupt circuits & clock circuits | Microcontroller is a powerful device includes all features of microprocessor on chip. It also has features of ROM, RAM, Parallel I/O, Serial I/O, timer |
| 2) Microprocessors includes many Operational codes for moving data from external memory to the CPU | Microcontrollers may have one or two |
| 3) Microprocessors may have one or two ~~bit~~ types of bits handling instructions | Microcontroller will have many |
| 4) Microprocessor must have many additional parts to be operational | Microcontroller functions as a computer with the addition of no external digital parts. |
| 5) Since memory & I/O has to be connected externally, the circuit becomes large | Since memory & I/O are present internally, the circuit is small. |
| 6) Mainly used in Personal Computers | mainly used in washing M/C, MP3 player |

# 8051 Microcontroller Architecture

## 8051 Microcontroller Hardware

The below figure shows a block diagram of 8051 microcontroller



8051 architecture consists of following features

→ Eight-bit CPU with registers A (The Accumulator) and B

→ Sixteen bit Program Counter (PC) and Data Pointer (DPTR)

→ Eight bit Program Status word (PSW)

→ Eight bit Stack Pointer (SP)

→ Internal Rom of 4k

→ Internal RAM of 128 bytes

→ Four register banks, each bank containing eight registers

→ Sixteen bytes which may be addressed at the bit level

→ Eighty bytes of general purpose data memory

→ Thirty two input/output pins arranged as four 8 bit ports: P0-P3

→ Two 16 bit Timer/Counters: T0 and T1

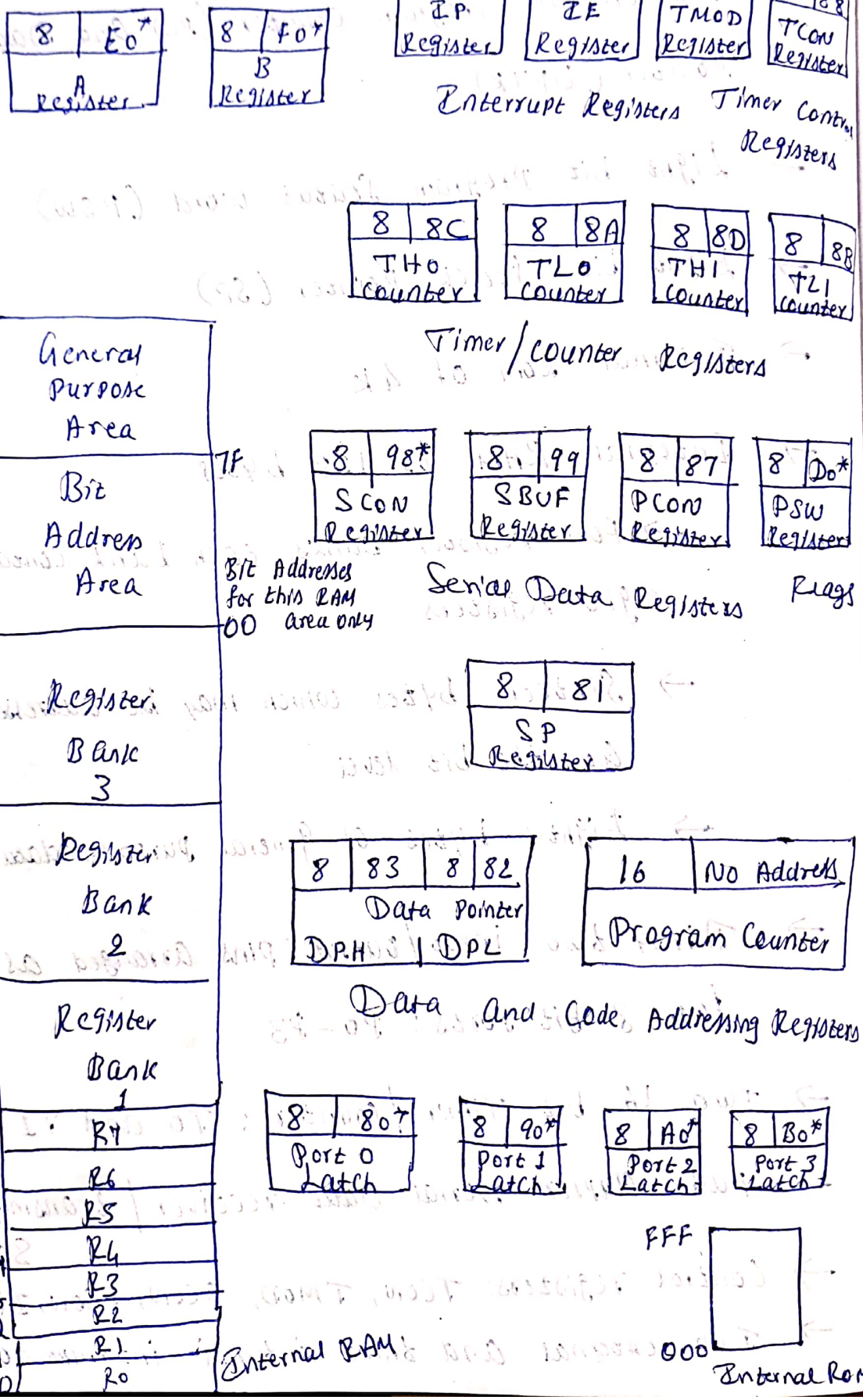
→ Full duplex Serial data receiver/transmitter: SBUF

→ Control registers: TCON, TMOD, SCON, PCON, IP & IE

→ Two external and three internal interrupt sources

→ Oscillator and Clock Circuits.

The programming model of 8051 is shown below.



## 1. A and B CPU Registers

The 8051 contains 34 general purpose registers. Two of these registers A and B hold results of many instructions, particularly math and logical operations of 8051 CPU. The other 32 general purpose registers are arranged as part of internal RAM as four banks B0 - B3, each bank with 8 registers.

The A (accumulator) register is the most versatile of two CPU registers, used for many operations including addition, subtraction, multiplication and division, and boolean bit manipulations. The B register is used with the A register for multiplication and division operations. The A register is also used for all data transfers between 8051 and external memory.

## 2. 16 bit Program Counter and Data Pointer

The 8051 contains two 16 bit registers: the Program Counter (PC) and Data Pointer (DPTR). Each is used to hold address of memory location.

Program ROM may be on the Chip at addresses 0000h to 0FFFh, external to the Chip for addresses that exceed 0FFFh or totally external for all addresses from 0000h to FFFFh. The PC is automatically incremented after fetching of each instruction or may also be altered by certain instructions (Jump instructions). The PC is only register that does not have any internal address.

DPTR register is made up of two 8 bit registers named DPH and DPL, which contains address of memory for code access or data access. DPTR can be specified in the program by its 16 bit name DPTR or by its individual byte name DPH and DPL. DPTR does not have single address, instead DPH and DPL each has individual address.

## Flags and 8 bit Program Status word (PSW)

Flags are 1 bit register provided to store status after the execution of instruction. In order that flags may be conveniently addressed, they

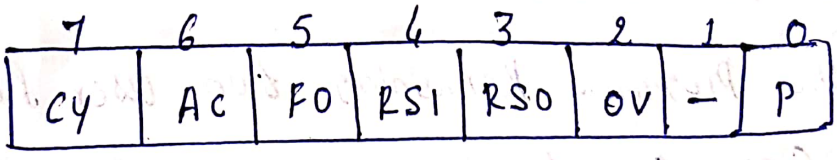
are grouped inside Program Status Word (PSW) and Power Control (PCON) registers.

8051 has four math flags and three General Purpose User flags. 4 math flags represents Status of the currently executed instruction. It includes

* Carry flag (C)
* Auxiliary Carry Flag (AC)
* Overflow Flag (OV)
* Parity Flag (P)

3 User flags are named as F0, GF0 & GF1, which can be set to 1 or cleared to 0 by the Programmer as desired.

Program Status word of 8051 microcontroller is as shown in below figure.




| Bit | Symbol | Function |
|---|---|---|
| 7 | CY | Carry flag; used in Arithmetic, Jump, rotate & boolean instns |
| 6 | AC | Auxiliary Carry flag; used for BCD arithmetic |
| 5 | F0 | |
| 5 | F0 | User flag 0 |

| | | |
|---|---|---|
| 4 | RS1 | Register bank Select bit 1 |
| 3 | RS0 | Register bank Select bit 0 |

| RS1 | RS0 | |
|---|---|---|
| 0 | 0 | Select Register bank 0 |
| 0 | 1 | Select Register bank 1 |
| 1 | 0 | Select Register bank 2 |
| 1 | 1 | Select Register bank 3 |

| | | |
|---|---|---|
| 2 | OV | Overflow flag; used in Arithmetic Instr |
| 1 | — | Reserved for future use |
| 0 | P | Parity flag; Shows Parity of Registe A ; 1 = Odd Parity |

PSW contains 4 maths flags, 1 user flag F0 and register select bits RS0 & RS1 used to select bits that identify which of the four General purpose register banks is currently in use by the program. Remaining two user flags GF0 and GF1 are stored in PCON register.

## Stack and Stack Pointer

Stack refers to an area of internal RAM. The 8-bit Stack Pointer (SP) register is used to

hold address of the top of the stack. When data is to be placed on the stack, SP increments and the stores data on to the stack, As data is retrieved from the stack, the byte is read from the stack, then SP decrements to point to next top of the stack data.

Stack is limited in height to the size of the internal RAM. The stack has the potential to overwrite valuable data in the register banks, bit-addressable RAM & Scratch pad RAM space.

## Internal RAM

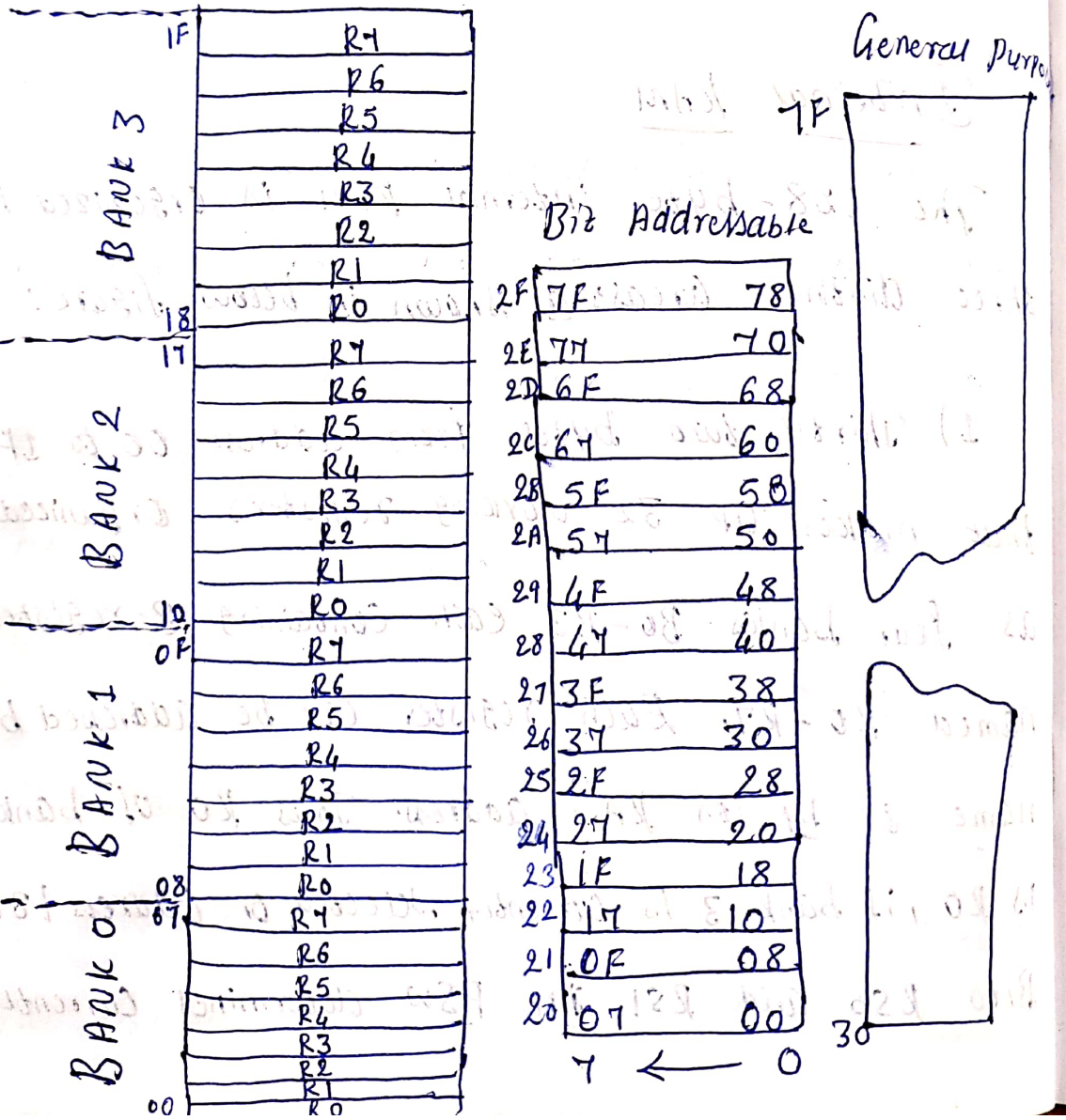
The 128-byte internal RAM is organized into three distinct areas as shown in below figure:

1) Thirty two bytes from address 00 to 1FH that makes up 32 working registers organized as four banks B0-B3, each containing 8 registers named R0-R7. Each register can be addressed by name or by its RAM address. Thus R0 of bank 3 is R0, if bank 3 is currently selected or address 18H. Bits RS0 and RS1 in PSW determines currently

using register bank. Register bank not current using can be used as General purpose RAM.

2) 16 bytes of bit addressable area of RAM addresses from 20h to 2Fh, forming a total of 128 addressable bits (16 * 8 = 128 bits). Addressable bits are useful when program need to handle binary event (switch on, light off etc).

3) A General purpose RAM area above from 30h to 7Fh.



# Input / Output Pins, Ports and Circuits

One major feature of a microcontroller is the versability built into the input/output (I/O) circuits that connect the 8051 to the outside world.

Microprocessor designs must add additional chips to interface with external circuitary, but this ability is built into the microcontroller.

8051 Microcontroller mainly has thirty two I/O pins grouped as four 8 bit ports: P0 - P3

## Port 0

Port 0: Pins may serve as inputs, outputs or as bidirectional low-order address and data bus for external memory.

## Port-1

Port 1 pins have no dual functions. It can be used either as input port or as an output port.

## Port 2

Port 2 may be used as an input/output port similar to port 1. Port 2 can also be used to send high-order address byte in conduction with port 0 lower-order byte to address external memory

## Port 3

Port 3 is an input/output port similar to port 1. Unlike Ports 0 and 2, which can have external addressing functions and change all eight port bits when in alternate use, each pin of Port 3 may be individually programmed to be used either as input/output or as one of the alternate functions as shown below

| Pin | Alternate use | SFR |
|---|---|---|
| P3.0 – RXD | Serial Data Input | SBUF |
| P3.1 – TXD | Serial Data output | SBUF |
| P3.2 – $\overline{INT0}$ | External Interrupt 0 | TCon.1 |
| P3.3 – $\overline{INT1}$ | External Interrupt 1 | TCon.3 |
| P3.4 – T0 | External timer 0 input | TMOD |
| P3.5 – T1 | External timer 1 input | TMOD |
| P3.6 – $\overline{WR}$ | External memory write pulse | – |
| P3.7 – $\overline{RD}$ | External memory Read pulse | – |

Rajeshwari BS
Assistant Professor
Department of CSE, BMS college of Engg

# Addressing modes Of 8051 Microcontroller

The different ways of specifying the address of source or destination in the mnemonic is called as addressing mode

8051 Microcontroller supports four addressing modes

* Immediate Addressing mode
* Register Addressing mode.
* Direct Addressing mode.
* Indirect Addressing mode.

## Immediate Addressing Mode

In Immediate address mode, the data is the part of the instruction itself.

The below figure shows immediate addressing mode.

| Instruction using # | Next Byte(s) are Data |
|---|---|

Immediate Addressing mode

The mnemonic for immediate data is the Pound sign (#).

Three mnemonics of immediate addressing modes are.

1) MOV Rr, #n ; Copy the 8 bit number n into register Rr

2) MOV A, #n ; Copy the 8 bit number n into the Accumulator register

3) MOV DPTR, #nn ; Copy the 16 bit number nn into the DPTR register.

Immediate addressing modes, using some of the registers from R0 to R7 in the currently available selected register bank, register A & register DPTR are shown below

MOV R0, #00h ; Put the immediate 8 bit number 00h in register R0

MOV R1, #01h ; Put the immediate 8 bit number 01h in register R1

MOV R4, #04h ; Put the immediate 8 bit number 04h in register R4

MOV R7, #07h ; Put the immediate 8 bit number 07 in register R7

MOV A, #0AAh ; Put the immediate 8 bit number AAh in register A

MOV DPTR, #1234h ; Put the immediate 16 bit numbe

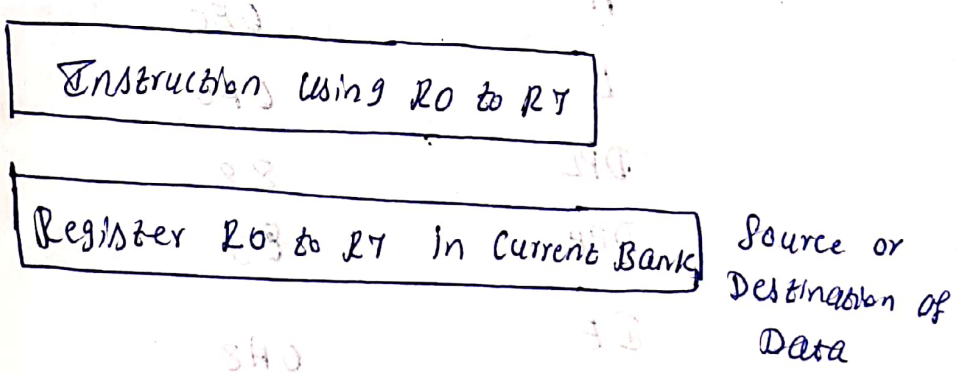
# Register Addressing Modes

Here, certain registers may be used as part of the instruction as sources or destinations of data. Registers A, DPTR and R0 to R7 may be used as part of instruction.

Register - to Register moves are as follows.

```
MOV A, Rr ; Copy data from register Rr to
MOV Rr, A ; register A
          ; Copy data from register A to
            register Rr.
```

The below figure shows register addressing mode.




A copy of data is made from the source to destination register.

Eg!

1) MOV A, R0 ; Copy the data in Register R0 to Register A

2) MOV R5, A ; Copy the data in register A to register R5

# Direct Addressing mode

All 128 bytes of internal RAM and the SFRs may be addressed by means of single byte address assigned to each RAM location and each special function register.

Internal RAM uses addresses from 00H to 7FH to address each byte. The SFR addresses exists from 80h to FFh.

The SFR addresses are shown below

| SFR | Address |
|---|---|
| A | 0E0 |
| B | 0F0 |
| DPL | 82 |
| DPH | 83 |
| IE | 0A8 |
| IP | 0B8 |
| P0 | 80 |
| P1 | 90 |
| P2 | 0A0 |
| P3 | 0B0 |
| PCON | 87 |
| PSW | 0D0 |

| | |
|---|---|
| SBUF | 99 |
| SCON | 98 |
| SP | 81 |
| TCON | 88 |
| TMOD | 89 |
| TH0 | 8C |
| TL0 | 8A |
| TH1 | 8D |
| TL1 | 8B |

Rajeshwari B S
Assistant Professor
Department of CSE
B.M.S College of Engg
Bengalure-19

RAM addresses 00 to 1Fh are assigned to 4 register banks, each with 8 working registers $R_0$-$R_7$.

For eg:- R2 of register bank 0 can be addressed as R2 in register addressing mode or in direct addressing mode as 02h. The direct addresses of 8 working registers of each bank is as shown below

| Bank | Register | Address |
|---|---|---|
| 0 | R0 | 00 |
| 0 | R1 | 01 |
| 0 | R2 | 02 |
| 0 | R3 | 03 |
| 0 | R4 | 04 |
| 0 | R5 | 05 |
| 0 | R6 | 06 |
| 0 | R7 | 07 |
| 1 | R0 | 08 |
| 1 | R1 | 09 |
| 1 | R2 | 0A |
| 1 | R3 | 0B |
| 1 | R4 | 0C |
| 1 | R5 | 0D |
| 1 | R6 | 0E |
| 1 | R7 | 0F |
| 2 | R0 | 10 |
| 2 | R1 | 11 |
| 2 | R2 | 12 |
| 2 | R3 | 13 |
| 2 | R4 | 14 |
| 2 | R5 | 15 |
| 2 | R6 | 16 |
| 2 | R7 | 17 |
| 3 | R0 | 18 |
| 3 | R1 | 19 |

| Bank | Register | Address |
|---|---|---|
| 3 | R2 | 1A |
| 3 | R3 | 1B |
| 3 | R4 | 1C |
| 3 | R5 | 1D |
| 3 | R6 | 1E |
| 3 | R7 | 1F |

Only one bank of working registers is active at any given time. The PSW special function register holds the bank select bits $RS_0$ & $RS_1$ which determines which register bank in use.

When 8051 is reset, $RS_0$ & $RS_1$ are set to 00 to select the working registers in bank 0, located from 00h to 07h in internal RAM. The Programmer may choose any other bank by setting RS0 and RS1 as desired.

Example for direct addressing mode

1) MOV A, 80h ; copy data from port 0 pins to register A

2) MOV 80h, A ; copy data from register A to the port 0

3) MOV 3Ah, #3Ah ; Copy immediate data byte 3Ah to RAM location 3Ah

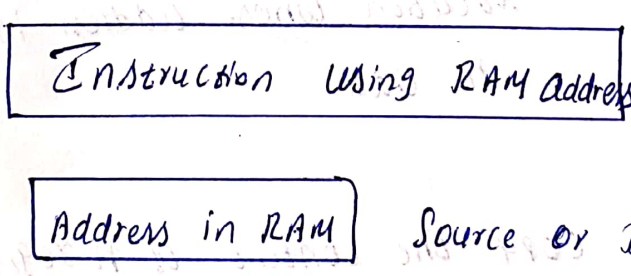
4) MOV R0, 12h ; Copy data from RAM location 12h to register R0

5) MOV 8Ch, R4 ; Copy data from register R4 to timer 0 high byte

6) MOV 5Ch, A ; Copy data from register A to RAM location 5Ch

7) MOV 0A8h, 7Fh ; Copy data from RAM location 7Fh to IE register.

The Pictorial representation of direct addressing mode is shown below




## Indirect Addressing Mode

Indirect addressing mode uses a register to hold address of data involved in data move. Indirect addressing mode for MOV instruction uses registers

R0 or R1 often called Data Pointer to hold the address of one of the location of RAM (Register)

The mnemonic symbol used for indirect addressing is @ symbol.

## Example for Indirect Addressing Mode

1) MOV A, @R0 ; Copy the contents of the address in R0 to A register

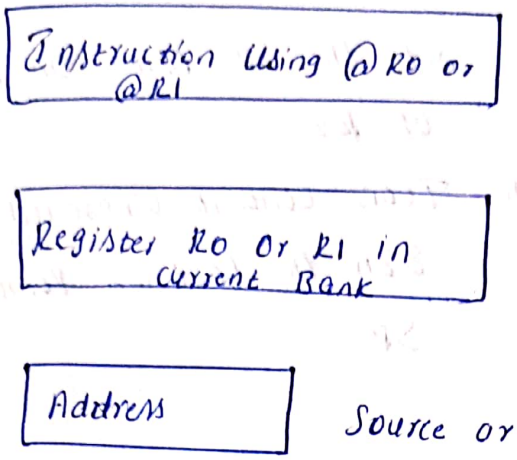
2) MOV @R1, #35h ; Copy the number 35h to the location whose address is in R1

3) MOV add, @R0 ; Copy the contents of the location whose address is in R0 to add

4) MOV @R1, A ; Copy the content of A register to the location whose address is in R1

5) MOV @R0, 80h ; Copy the contents of the Port 0 pins to the location whose address is in R0.

The pictorical representation of Indirect Addressing mode is shown below




Example Programs / Problems

1) Copy the byte in TCON to register R2 using atleast four different methods

* Method 1: Use the direct address for TCON (88h) and register R2:

```
Mov R2, 88h ; Copy TCON content to R2
```

* Method 2: Use direct address for both TCON & R2

```
Mov 02h, 88h ; Copy TCON content to register R2 whose address is 02h
```

* Method 3:- Use R1 as a pointer to R2 (indirect addressing)

```
Mov R1, #02h ; Move address of R2 to R1
Mov @R1, 88h ; Copy TCON content to location pointed by R1
```

* Method 4 ; Push the contents of TCON into direct address 02h ie, R2

```
MOV 81h, #02h ; Set the SP with the address of R2
Push 88h ; Push, TCON content whose address is 88h to location pointed by SP.
```

2) Set timer T0 to an initial setting of 1234h

```
MOV 8Ch, #12h ; Set TH0 to 12h
MOV 8Ah, #34h ; Set TL0 to 34h
```

3) Put the number 34h in registers R5, R6 and R7

Method-1 ; Using Immediate addressing mode

```
MOV R5, #34h ; move 34 to R5
MOV R6, #34h ; move 34 to R6
MOV R7, #34h ; move 34 to R7
```

Method-2 ; Put the number in A & MOV A to each register

```
MOV A, #34h ; Move 34h to A
MOV R5, A ; Move Content of A to R5
MOV R6, A ; Move content of B to R6
MOV R7, A ; Move content of C to R7
```

Method-3: Copy Using direct addressing mode

```
MOV R5, #34h ; Move 34h to Register R5
MOV 06h, 05h ; Move content of R5 whose address is
               05h to register R6 whose address is
               06h
MOV 07h, 06h ; Move content of R6 whose address
               is 06h to register R7 whose
               address is 07h.
```

4) Put the number 8Dh in RAM locations 30h to 34h

Method 1: Using Immediate addressing mode

```
MOV 30h, #8Dh ; Copy the number 8Dh to location
                30h
MOV 31h, #8Dh; Copy the number 8Dh to location 31h
MOV 32h, #8Dh ; Copy the number 8Dh to location 32h
MOV 33h, #8Dh ; Copy the number 8Dh to location 33h
MOV 34h, #8Dh; Copy the number 8Dh to location 34h
```

Method-2 ; Using Register addressing mode

```
MOV A, #8Dh ; Copy the number 8Dh to A
MOV 30h, A ; Copy Content of A register to location
             30h
MOV 31h, A ; Copy Content of A register to location 31h
MOV 32h, A ; Copy Content of A register to location 32h
MOV 33h, A ; Copy Content of A register to location 33h
MOV 34h, A ; Copy Content of A register to location 34h
```

Method-3: Using PUSH Instruction.

```
MOV 30h, #8Dh; Copy the number 8Dh to RAM location
               30h
```

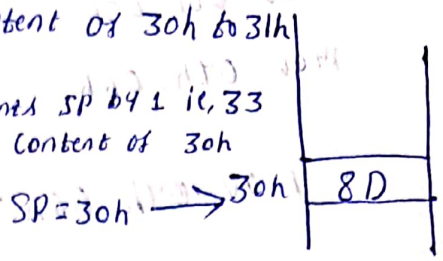
MOV 81h, #30h ; Set SP to location 30h

PUSH 30h ; Increments SP by 1 ie, 31h & Push Content of 30h to 31h

Increments SP by 1 ie, 32 & Push Content of 30h ; PUSH 30h

Increments SP by 1 ie, 33 & Push Content of 30h ; PUSH 30h

Increments SP by 1 ie, 34 & Push content of 30h ; PUSH 30h




5) Swap the contents of Registers R7 and R6 in register bank 0.

Method-1 Using Register addressing mode

```
MOV A, R6  ; Copy content of R6 to A
MOV R5, A  ; Temporarily save it in R5
MOV A, R7  ; Copy content of R7 to A
MOV R6, A  ; Copy content of A ie, R7 to R6
MOV A, R5  ; Copy temporarily saved content in R5 to A
MOV R7, A  ; Copy content of A ie, R6 to R7
```

Method-2 Using direct addressing mode

```
MOV 10h, 06h ; Copy content of R6 to RAM location 10h
MOV 06, 07h  ; Copy content of R7 whose address is 07 to R6 whose address is 06
```

MOV 07, 10h ; Copy saved content from location 10h to R7 whose address is 07h

Method-3 Using PUSH & POP Instructions

PUSH 07H ; Push content of R7 onto stack

PUSH 06H ; Push contents of R6 onto stack

POP 07H ; Pop contents of R6 to R7

POP 06H ; Pop content of R7 to R6

Method-4 Using Exchange Instructions

XCH A, R6 ; Exchange content of R6 & A

XCH A, R7 ; Exchange content of R7 & A
ie, content of R6 now in R7

XCH A, R6 ; Exchange content of R6 & A
ie, content of R7 now in R6

=

# Logical Operations

8051 microcontroller supports logical operations at two data levels; Byte Level or Bit Level.

Logical operations supported by 8051 microcontroller & corresponding mnemonics is as shown below

| Logical Operation | 8051 Mnemonic |
|---|---|
| AND | ANL (Logical AND) |
| OR | ORL (Logical OR) |
| XOR | XRL (Logical Exclusive OR) |
| NOT | CPL (Complement) |

The source can be either immediate, register, direct address, Indirect address or register A. The destination can be either register A or direct address. Logical operator performs operations operations on each individual bit of source data and destination data. This operation is called as Byte-Level Logical operation because the entire byte is affected.

The Byte-Level Logical Operations supported by 8051 Microcontroller are as follows

## AND Instructions

1) ANL A, #n ; AND each bit of A with each bit of immediate number n and store the result in A

2) ANL A, add ; AND each bit of A with each bit of a number stored in RAM location and store the result in A.

3) ANL A, Rr ; AND each bit of A with each bit of a number stored in register Rr & store the result in A

4) ANL A, @Rp ; AND each bit of A with each bit of a number stored in RAM location whose address is in Rp and put the result in A

5) ANL add, A ; AND each bit of A with each bit of a number stored in RAM location and store the result in RAM location.

6) ANL add, #n ; AND each bit of a number n with each bit of a number stored in RAM location and store the result in RAM location.

## OR Instructions

1) ORL A, #n ; OR each bit of A with each bit of a number and store the result in A

2) ORL A, add ; OR each bit of A with each bit of a number stored in RAM location and store the result in A.

3) ORL A, $R_r$ ; OR each bit of A with each bit of a number stored in register $R_r$ and store the result in A.

4) ORL A, @$R_p$ ; OR each bit of A with each bit of a number stored in RAM location whose address is in $R_p$ and store the result in A

5) ORL add, A ; OR each bit of A with each bit of a number stored in RAM location and store the result in RAM location

6) ORL add, #n ; OR each bit of a number n with each bit of a number stored in RAM location and store the result in RAM location.

# XOR - Instructions

1) XRL A, #n ; XOR each bit of A with each bit of a number n and stores the result in A

2) XRL A, add; XOR each bit of A with each bit of a number stored in RAM location & stores the result in A

3) XRL A, $R_r$ ; XOR each bit of A with each bit of a number stored in register and stores the result in A

4) XRL A, @$R_p$; XOR each bit of A with each bit of a number stored in RAM location whose address is in $R_p$ and stores the result in A.

5) XRL add, A ; XOR each bit of A with each bit of a number stored in RAM location and stores the result in RAM location

6) XRL add, #n ; XOR each bit of immediate number n with each bit of a number stored in RAM location and stores the result in RAM location.

## NOT Instructions

1) CLR A; Clear each bit of A to 0

2) CPL A; Complement each bit of A; makes bit 0 as 0 and bit 0 as 1

## Eg: for Logical Instructions

| | |
|---|---|
| MOV A, #0FFh | A=FFh |
| MOV R0, #77h | R0=77h |
| ANL A, R0 | A=77h |
| MOV 15h, A | 15h=77h |
| CPL A | A=88h |
| ORL 15h, #88h | 15h=FFh |
| XRL A, 15h | A=77h |
| XRL A, R0 | A=00h |
| ANL A, 15h | A=00h |
| ORL A, R0 | A=77h |
| CLR A | A=00h |
| XRL 15h, A | 15h=FFh |
| XRL A, R0 | A=77h |

# Bit-Level Logical Operation

Bit-level logical operation is very convenient when we wish to alter a single bit of a byte. RAM locations 20h to 2Fh supports for both byte level operations and bit level operations. 20h to 2Fh locations ie, 16 bytes are called bit addressable locations with each location 8 bits. Thus the bit addresses are numbered from 00 to 7Fh ie. 00-128d (16 bytes * 8 bit = 128 bits)

Bit 0 of a byte location 20h has address 00h

Bit 1 of a byte location 20h has address 01h

Bit 7 of a byte location 20h has address 07h

Bit 0 of a byte location 21h has address 08h

The bit addressable locations and corresponding bit address is shown below

| Byte address | | Bit address |
|---|---|---|
| 20h | ⟶ | 00-07h |
| 21h | ⟶ | 08-0F |
| 22h | ⟶ | 10-17 |
| 23h | ⟶ | 18-1F |
| 24h | ⟶ | 20-27 |
| 25h | ⟶ | 28-2F |
| 26h | ⟶ | 30-37 |

27 ⟶ 38-3F

28 ⟶ 40-47

29 ⟶ 48-4F

2A ⟶ 50-57

2B ⟶ 58-5F

2C ⟶ 60-67

2D ⟶ 68-6F

2E ⟶ 70-77

2F ⟶ 78-7F

SFR and the corresponding bit addresses are shown below

| SFR | Direct Address | Bit Addressable |
|---|---|---|
| A | 0E0 | 0E0-0E7 |
| B | 0F0 | 0F0-0F7 |
| IE | 0A8 | 0A8-0AF |
| IP | 0B8 | 0B8-0BF |
| P0 | 80 | 80-87 |
| P1 | 90 | 90-97 |
| P2 | 0A0 | 0A0-0A7 |
| P3 | 0B0 | 0B0-0B7 |

| | | |
|---|---|---|
| PSW | ODO | ODO-ODF |
| TCON | 88 | 88-8F |
| SCON | 98 | 98-9F |

Raicshwari BS
Assistant Professor
Department of CSE
B.M.S College of Engg
Bengaluru

For eg:

To clear A register, we write

CLA A

To clear 5th bit of the register A, the instruction is

CLR ACC.5

Bit-Level Logical Operations are as follows:

1) ANL C,b ; AND Carry flag C and addressed bit and store the result in carry flag C

2) ANL C,/b ; AND C and the complement of addressed bit and store the result in carry flag C Addressed bit is not altered.

3) ORL C,b ; OR carry flag C and addressed bit & store the result in carry flag C

4) ORL C,/b ; OR Carry flag C and the complement of the addressed bit and stores the result in C. The addressed bit is not altered.

5) CPL C ; Complement the carry flag content

6) CPL b ; Complement the addressed bit

7) CLR C ; Clear the carry flag content to 0

8) CLR b ; Clear the addressed bit to 0

9) MOV C, b ; Copy the addressed bit to carry flag

10) MOV b, C ; Copy the carry flag content to the addressed bit

11) SETB C ; Set the carry flag content to 1

12) SETB b ; Set the addressed bit to 1

# Rotate and Swap Operations

The register A can be rotated one bit position to the left or right with or without including the C flag (carry flag) in the rotation. If the C flag is not included, then the rotation involves eight bits of register A. If the C flag is included, then nine bits are involved in the rotation.

The various rotation instructions supported by 8051 Microcontroller are:

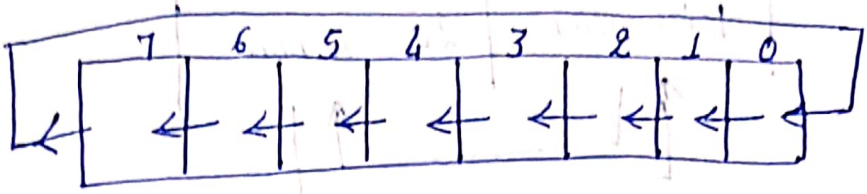
1) RL A ; Rotate the content of A register by one bit position to the left;

2) RLC A ; Rotate the content of A register and the carry flag content one bit position to the left.

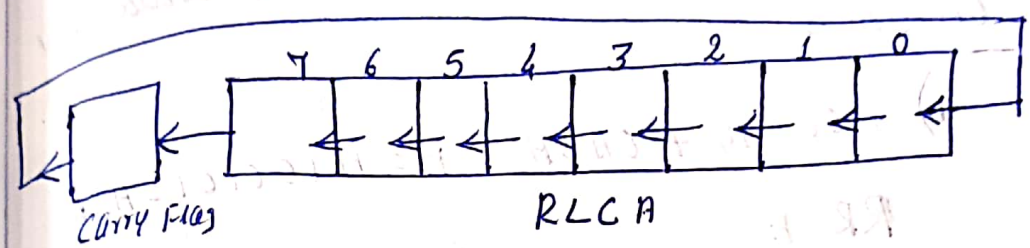
3) RR A; Rotate the content of A register by one bit position to the right.

4) RRC A; Rotate the content of A register and carry flag by one bit position to the right

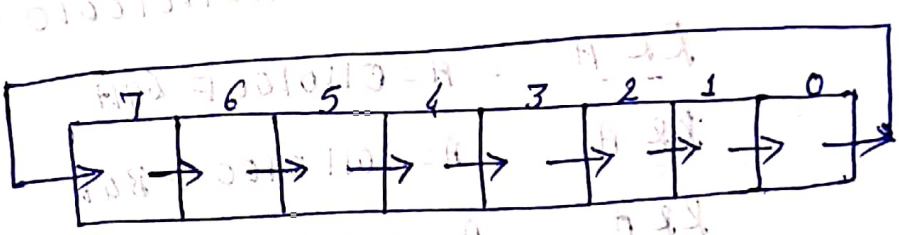
The diagramatic representation of various rotate instructions is shown below.



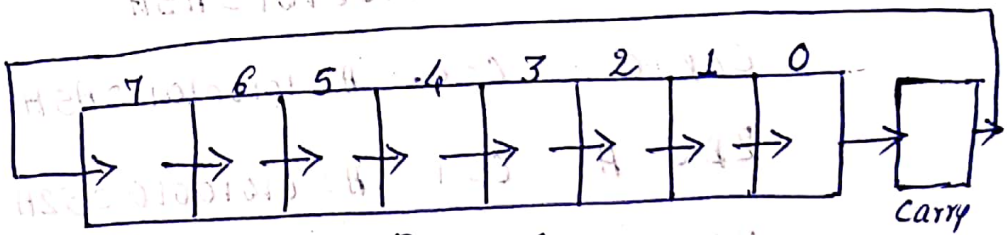

RL A




RLC A




RR A




RRC A

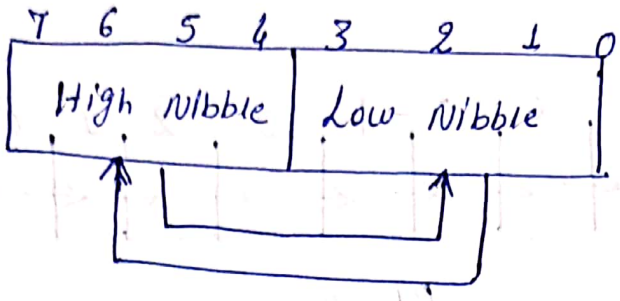
No flags other than the carry flag in RRC & RLC are affected

## SWAP Instruction

SWAP A; Interchange the nibbles of register A; Put high nibble in the low nibble position and low nibble in the high nibble position.

The diagramatic representation of SWAP instruction is shown below

1)




The following list shows examples of rotate and swap operations

| | |
|---|---|
| MOV A, #0A5H | A = 10100101 = A5H |
| RR A | A = 11010010 = D2H |
| RR A | A = 01101001 = 69H |
| RR A | A = 10110100 = B4H |
| RR A | A = 01011010 = 5AH |
| SWAP A | A = 10100101 = A5H |
| CLR. C | C = 0 ; A = 10100101 = A5H |
| RRC A | C = 1 A = 01010010 = 52H |
| RRC A | C = 0 A = 10101001 = A9H |
| RL A | A = 01010011 = 53H |
| RL A | A = 10100110 = A6H |
| SWAP A | C = 0 A = 01101010 = 6AH |
| RLC A | C = 0 A = 11010100 = D4H |
| RLC A | C = 1 A = 10101000 = A8H |
| SWAP A | C = 1 A = 10001010 = 8AH |

2)

b)

## Example Problems

1) Use three different Instructions to clear the contents of A register

Method-1: Using MOV Instruction

```
MOV A, #00H
```

Method-2; Using CLR Instruction

```
CLR A
```

Method-3: Using ANL Instruction

```
ANL A, #00H
```

2) Write a program that will invert (complement) every bit in register R6 of bank 0

Method 1:

```
MOV A, R6
CPL A
MOV R6, A
```

Method 2:

```
MOV A, #0FFH
XRL A, R6
MOV R6, A
```

Method-3

```
XRL 06H, #0FFH
```

3) Configure Timer T0 to act as a 16 bit timer and Timer T1 to act as a 13 bit counter and then start them

```
SETB ACC.0  ; T0 timer mode is 1; M0(T0)=1
CLR ACC.1   ; M1(T0)=0
CLR ACC.2   ; C/T(T0)=0 to time
CLR ACC.3   ; Disable T0 gate bit
CLR ACC.4   ; T1 timer mode is 0; M0(T1)=0
CLR ACC.5   ; M1(T1)=0
SETB ACC.6  ; C/T(T1)=1 to count
CLR ACC.7   ; Disable T1 gate bit
MOV TMOD, A ; configure timers
SETB TR1    ; Set T1 run bit
SETB TR0    ; Set T0 run bit
MOV TMOD, #41H ; Also configures the timers
```

4) Set and then clear bit 7 of internal RAM address 28H without changing any other bits in the byte

```
ORL 28H, #80H ; 1 OR X=1; Set bit 7 only
ANL 28H, #7FH ; 0 AND X=0; Clear bit 7 only
SETB 47H ; set the bit
CLR 47H  ; clear the bit
```

40H represents bit 0 of bit addressable location 28H

5) Double the number in register R2 and put the result in registers R3 (high byte) and R4 (low byte)

```
CLR C      ; Clear the carry to rotate as MSB of byte.
MOV A, R2  ; Get R2 to A
RLC A      ; Rotate left, which doubles the number
MOV R4, A  ; Put low byte of result in R4
CLR A      ; Clear A to receive carry
RLC A      ; The carry bit is now bit 0 of A
MOV R3, A  Transfer any carry bit to R3
```

5) OR the contents of Ports 1 and 2; put the result in external RAM location 0100H

```
MOV A, 90H  ; Copy the Pin data from PORT 1 to A
ORL A, 0A0H ; OR the contents of A and Port 2
MOV DPTR, #0100H ; Set DPTR with external RAM
                   address 0100H
MOV @DPTR, A ; Store the result.
```

6) Find a number that, when XORed to the A register, results in the number 3FH in A

```
MOV R0, A   ; Save A in R0
XRL A, #3FH XOR A and 3FH; Forming N
XRL A, R0   ; XOR A and N yielding 3FH
```

Eg: A=FFH  A XOR 3FH = C0H & C0H XOR FFH = 3FH

8) Swap every even numbered bit of Register R3 in bank 0 with the odd numbered bit to its left. Swap bit 0 with bit 1, bit 2 with bit 3, and so on until bit 6 is swapped with bit 7.

```
MOV  A, R3;  Copy R3 Content to A
RL A ;  Even-numbered bits to Odd numbered
                Positions
ANL A, #0AAH;  Mask or Set the odd numbered
                bits to 0
Push ACC  ;  Save even numbered bits
MOV A, R3  ;  Copy R3 to A
RL A  ;  Odd numbered bit to even numbered
                Positions
ANL A, #055H;  Mask out the old odd numbered
                Positions
MOV R3, A  ;  Save odd numbered bits
POP ACC  ;  Retrieve even numbered bits
ORL 03H, A;  OR the even and odd numbered bits
                into R3
```

# Arithmetic Instructions

24 Arithmetic opcodes are supported by 8051 microcontroller which are grouped into following types

1) INC Destination ; Increment destination by 1

2) DEC Destination ; Decrement destination by 1

3) ADD / ADDC Destination, Source ; Add Source to destination without / with Carry flag

4) SUBB destination, Source ; Subtract with Carry Source from destination

5) MUL AB ; Multiple the contents of A and B registers

6) DIV AB ; Divide the contents of register A by the Content of register B

7) DA A ; Decimal Adjust the A register

## Incrementing and Decrementing

Increment Instruction increments the specified register content or memory location content by 1 and decrement instruction decrements the specified register Content or memory location content by 1. The

Increment and decrement instruction does not affects the math flags: C, AC, OV.

The following are the mnemonics for increment and decrement instructions

INC A; Increments A register content by 1

INC $R_r$; Increments specified register content $R_r$ by 1

INC add; Increments specified memory location content by 1

INC @$R_p$; Increments content of a memory location pointed by register $R_p$ (whose address is in register $R_p$)

INC DPTR; Increments DPTR content by 1

DEC A; Decrements specified A register content by 1

DEC $R_r$; Decrements specified register content $R_r$ by 1

DEC add; Decrements specified memory location content by 1

DEC @$R_p$; Decrements memory location content by 1 whose address is in $R_p$.

## Addition

In 8051 microcontroller, all addition is done with A register and A register is the destination of the result. Source can be either immediate, Register, direct or indirect address.

The mnemonics of addition instructions are

ADD A, #n ; Add A and immediate number n & Store the result in A

ADD A, $R_y$ ; Add A and register content $R_y$ and Store the result in A

ADD A, add ; Add A and memory location content add & Store the result in A

ADD A, @$R_p$ ; Add A and memory location content Pointed by $R_p$ and Store the result in A

C flag is set to 1, if there is a carry out of 7Th bit otherwise cleared to 0

Ac flag is set to 1, if there is a carry out of 3rd bit, otherwise cleared to 0

OV flag set to 1, if there is a carry out of 7th bit position but not out of 6th bit position or if carry out of 6th bit position but not out of 7th bit position.

## Unsigned Addition

Unsigned numbers make use of the carry flag to detect when the result of ADD operation is larger than FFH. If the carry is set to 1 after an ADD then the carry can be added to a higher order by

Eg:

$$95D = 01011111$$
$$\underline{189D = 10111101}$$
$$284D \quad 1)00011100 = 284D$$

Where ) indicates the status of C flag.

The Program could add carry flag to second by of larger number.

## Signed Addition

Signed numbers may be added two ways

i) Addition of like signed numbers

ii) Addition of unlike signed numbers

If unlike signed numbers are added, then the result will not be larger than -128D to +127

Eg:

$-001D = 11111111$

$+027D \quad 00011011$

$+026D \quad 1)\ 00011010 = +026D$

Here, there is a carry from bit 7 so the carry flag is 1. There is also carry from bit 6, and the OV flag is 0. For this condition, no action need to be taken by the program to correct the sum.

Case-2:

If positive numbers are added, there is the possibility that the sum will exceed +127d as demonstrated in below example

$+100D = 01100100$

$+050D = 00110010$

$+150D \quad 0)\ 10010110 = \cancel{-150D}\ -106D$

Ignoring the sign bit. There is no carry from bit 7 & carry flag is 0 and there is a carry from bit 6 so the OV flag is 1.

Case 3: An example of adding two positive numbers that do not exceed positive limit

$+045D = 00101101$

$+075D = 01001011$

$+120D \quad 0)\ 01111000 = 120D$

There is no carry from bits 6 or 7; The carry and OV

flags are both 0.

Case 4: The result of adding two negative numbers together for a sum that does not exceed the negative limit is demonstrated below

$$\begin{array}{ll} -30D & = 11100010 \\ \underline{-050D} & = \underline{11001110} \\ -080D & 1)\ 10110000 = -80D \end{array}$$

Here, there is a carry from bit 7 and carry flag is 1. There is a carry from bit 6 and OV flag is 0. These are the same flags as the case of adding unlike numbers. No corrections are needed for the sum.

Case 5: When adding two negative numbers whose sum does not exceed $-128D$

$$\begin{array}{ll} -070D & 10111010 \\ \underline{-070D} & \underline{10111010} \\ -140D & 1)\ 01110100 = +116D \end{array}$$

There is a carry from bit position 7 and carry from bit position 6. So the carry and the OV flags are set to 1. The magnitude of the sum is correct; the sign bit must be changed to a 1.

Take 2's complement of the result 01110100

$$\begin{array}{r} 10001011 \\ \underline{\quad\quad\quad 1} \\ 10001100 = -140D \end{array}$$

Generally the following actions need to be taken

| C | OV | Action |
|---|---|---|
| 0 | 0 | None |
| 0 | 1 | Complement the sign |
| 1 | 0 | None |
| 1 | 1 | Complement the sign |

## Multibyte Signed Arithmetic

The mnemonics for Add with Carry are:

1) ADDC A, #n ; Add the content of A, Immediate number n and C flag and put the result in A

2) ADDC A, add ; Add the content of A, memory location content add and C flag, Store the result in A

3) ADDC A, $R_r$ ; Add the content of A, register cont. $R_r$ and C flag, Store the result in A

4) ADDC A, @$R_p$ ; Add the content of A, memory locat. Content Pointed by $R_p$, Carry flag & Store the result in A

Eg: Examples of ADD & ADDC multi-byte signed arithmetic operations

```
MOV  A, #1CH     A=1CH
MOV  R5, #0A1H   R5=A1H
ADD  A, R5  ;    A=BDH ; C=0, OV=0
ADD  A, R5  ;    A=5EH ; C=1 OV=1
ADDC A, #10H ;   A=6FH  C=0, OV=0
ADDC A, #10H ;   A=7FH, C=0, OV=0
```

Subtraction

# Multiplication and Division

The 8051 has the capability to perform 8-bit integer multiplication and division using the A and B registers.

The Register A holds 1 byte of data before a multiplication or division and 1 byte of the result after multiplication or division operation.

Multiplication and division treat the numbers in register A and B as unsigned.

## Multiplication

The multiplication instruction multiplies unsigned number in Register A by the unsigned number in register B and low order byte of the product is stored in A and high order byte of the product is stored in B.

Multiplication operation uses registers A and B as both source and destination.

MUL AB ; Multiply A by B, store the low-order byte of the product in A and high order byte of the product in B

The OV flag will be set if A X B > FFH, Signals the number is larger than 8 bits & the programmer need to consider register B content for high-order byte of the product.

Rajeshwari B S
Assistant Professor
Department of CSE
B.M.S College of Engg
Bengaluru-19

Eg:-

```
MOV A, #7BH ; A=7BH
MOV 0F0H, #02H  B = 02H
MUL AB          A=F6H and B=00H OV=0
MOV B, #0FE ; B=FEH
MUL AB ; A=14H and B=F4H OV=1
```

Division

Division instruction divides the unsigned number in register A by unsigned number in register B. Division Operation uses ~~both~~ registers A and B as both source as well as destination

DIV AB ; Divide A by B ; Stores integer part of the Quotient in register A and integer part of the remainder in B

Division Operation always results in integer Quotients and remainders as shown below

$$\frac{A = 213D}{B = 017D} = 12 \text{ (Quotient) and } 9 \text{ (Remainder)}$$