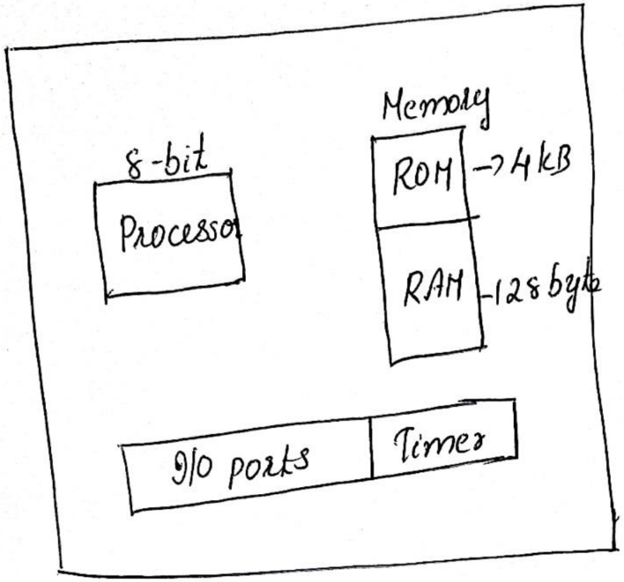
8051 MICROCONTROLLER

What is a microcontroller?

A microcontroller is a small, low-cost computer-on a-chip which usually includes:

- An 8 or 16 bit (CPU).
- A small amount of RAM.
- Programmable ROM and/or flash memory.
- Parallel and/or serial I/O.
- Timers and signal generators.
- Analog to Digital (A/D) and/or Digital to Analog (D/A) conversion.

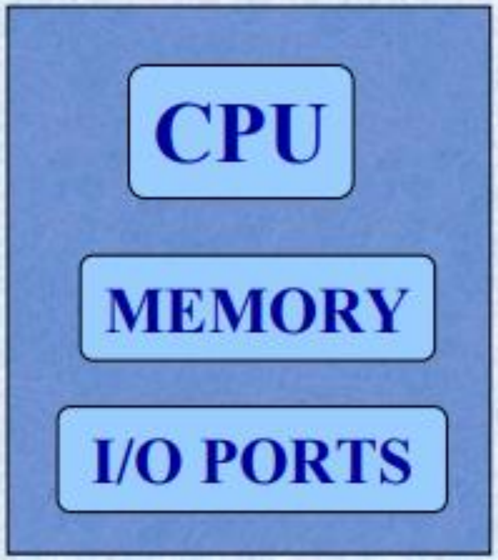


ROM - Program m/m
RAM - Data m/m

Difference

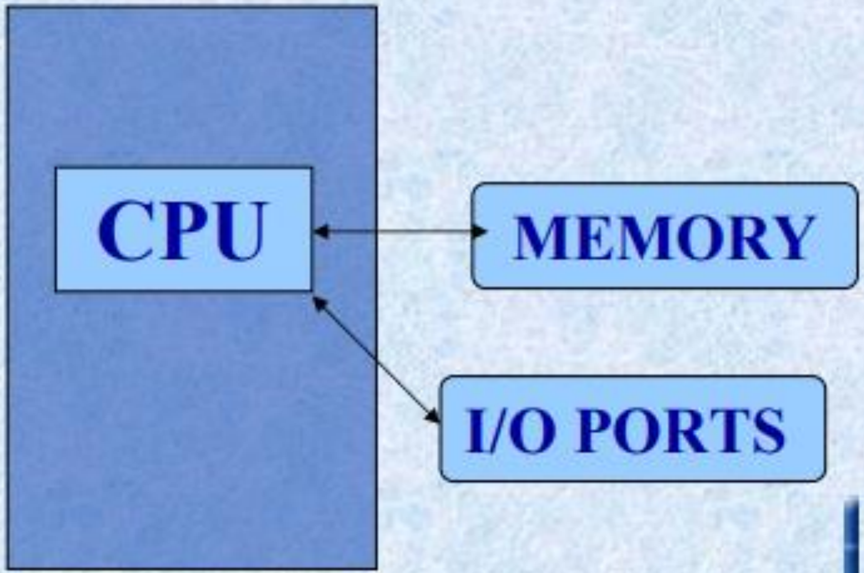
MICRO CONTROLLER

- It is a single chip
- Consists Memory,
- I/o ports

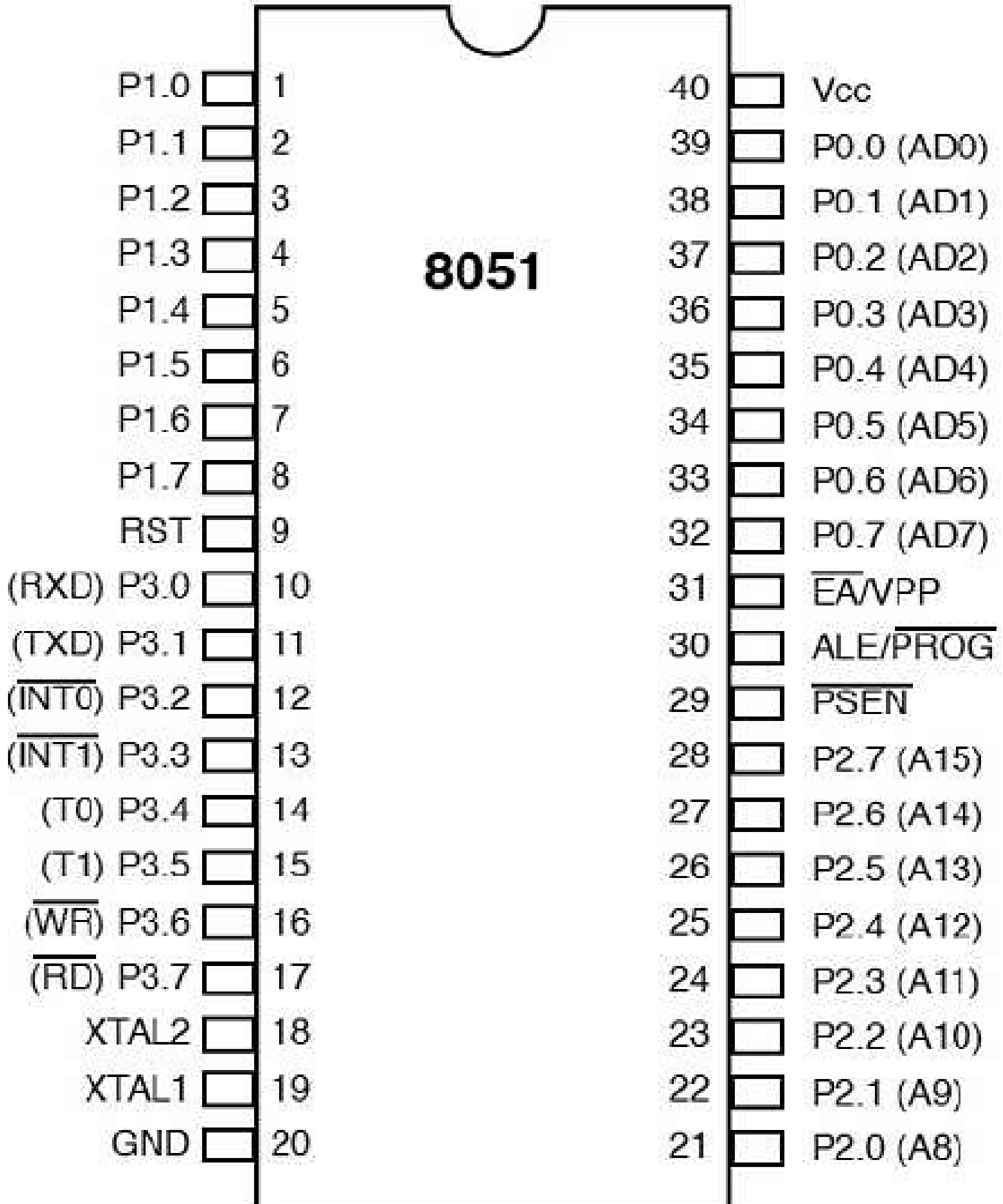


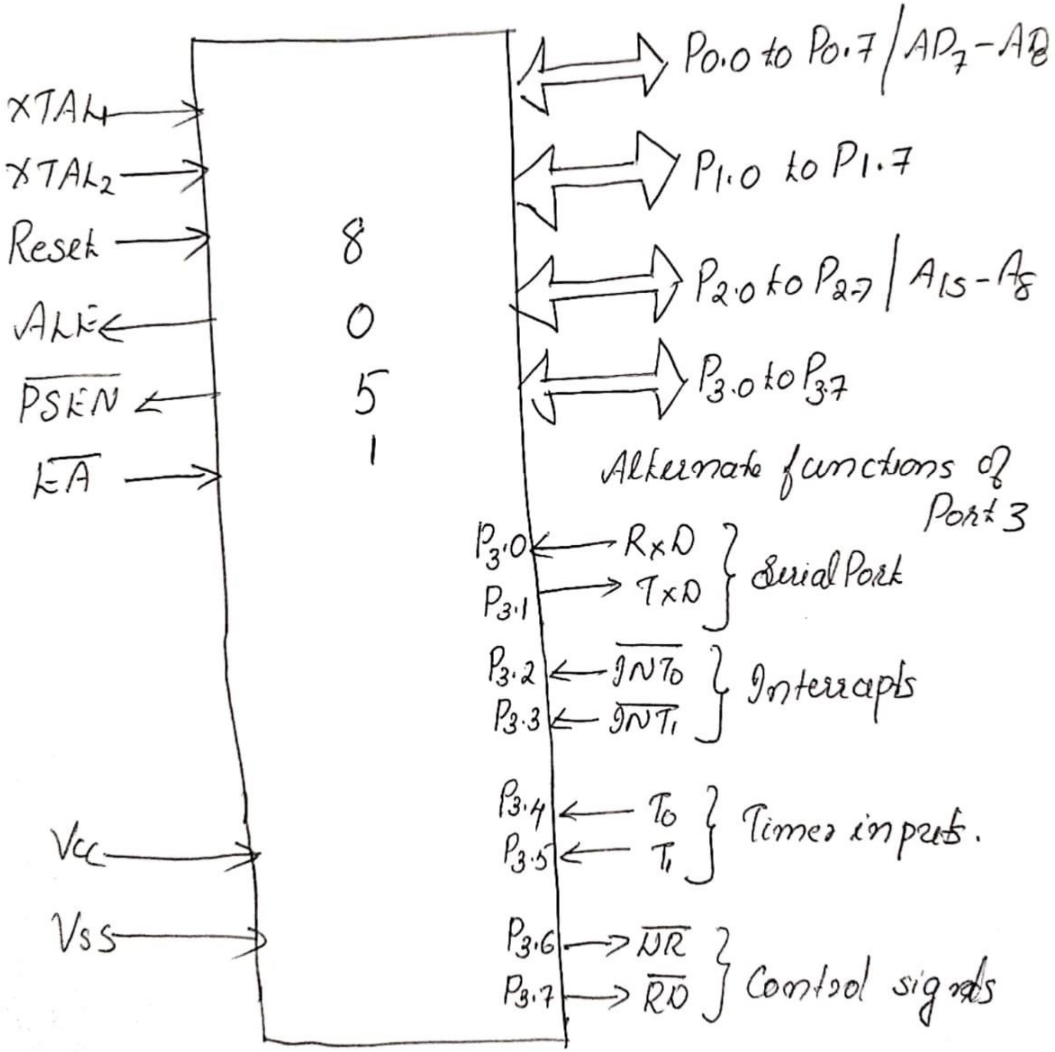
MICRO PROCESSER

- It is a cpu
- Memory, I/O Ports to be connected externally



Pin Diagram of 8051




XTAL1
XTAL2
Reset
ALE
$\overline{PSEN}$
$\overline{EA}$
Vcc
Vss
8
0
5
1
P0.0 to P0.7 / AD7-AD0
P1.0 to P1.7
P2.0 to P2.7 / A15-A8
P3.0 to P3.7
Alternate functions of Port 3
P3.0
RxD
P3.1
TxD
Serial Port
P3.2
$\overline{INT_0}$
P3.3
$\overline{INT_1}$
Interrupts
P3.4
T0
P3.5
T1
Timer inputs.
P3.6
$\overline{WR}$
P3.7
$\overline{RD}$
Control signals

Pins 1 to 8 − These pins are known as Port 1. This port doesn’t serve any other functions. It is internally pulled up, bi-directional I/O port.

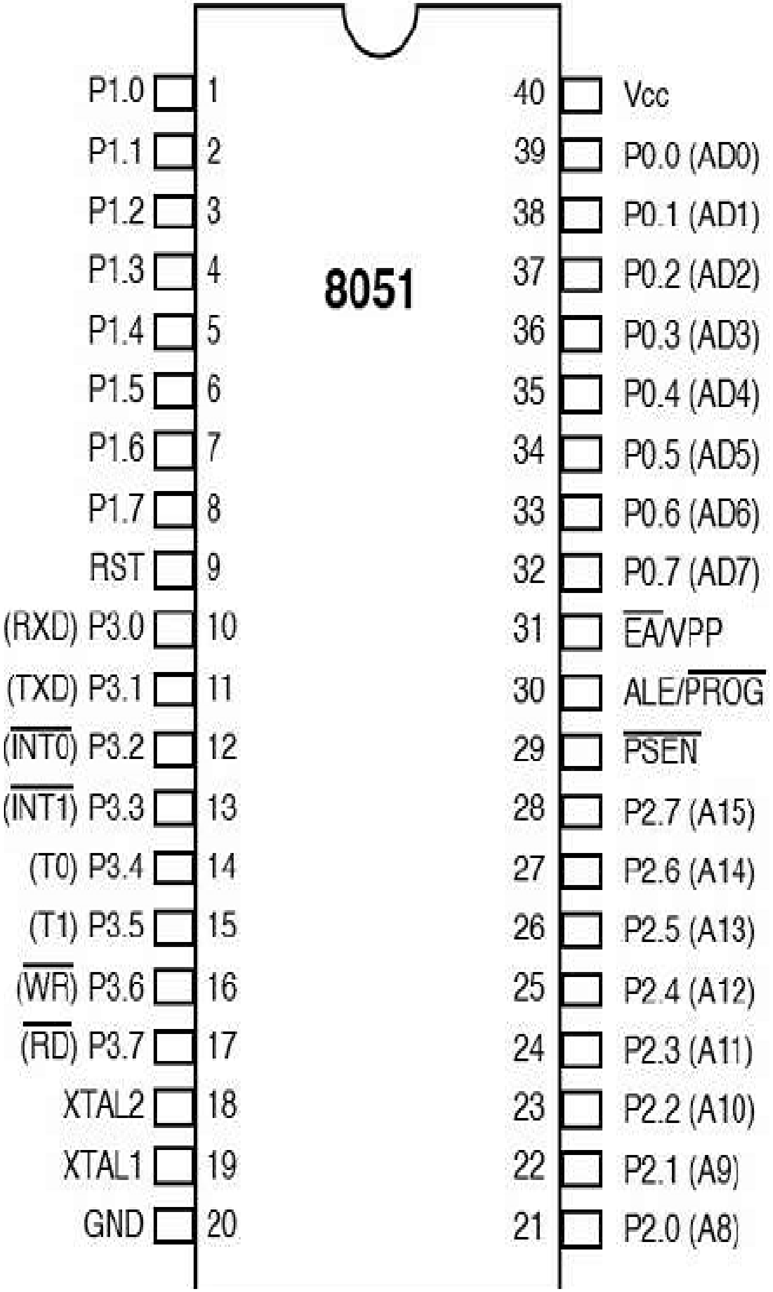
Pin 9 − It is a RESET pin, which is used to reset the microcontroller to its initial values.

Pins 10 to 17 − These pins are known as Port 3. This port serves some functions like interrupts, timer input, control signals, serial communication signals RxD and TxD, etc.

Pins 18 & 19 − These pins are used for interfacing an external crystal to get the system clock.

Pin 20 − This pin provides the ground supply to the circuit.

Pins 21 to 28 − These pins are known as Port 2. It serves as I/O port. Higher order address bus signals are also multiplexed using this port.



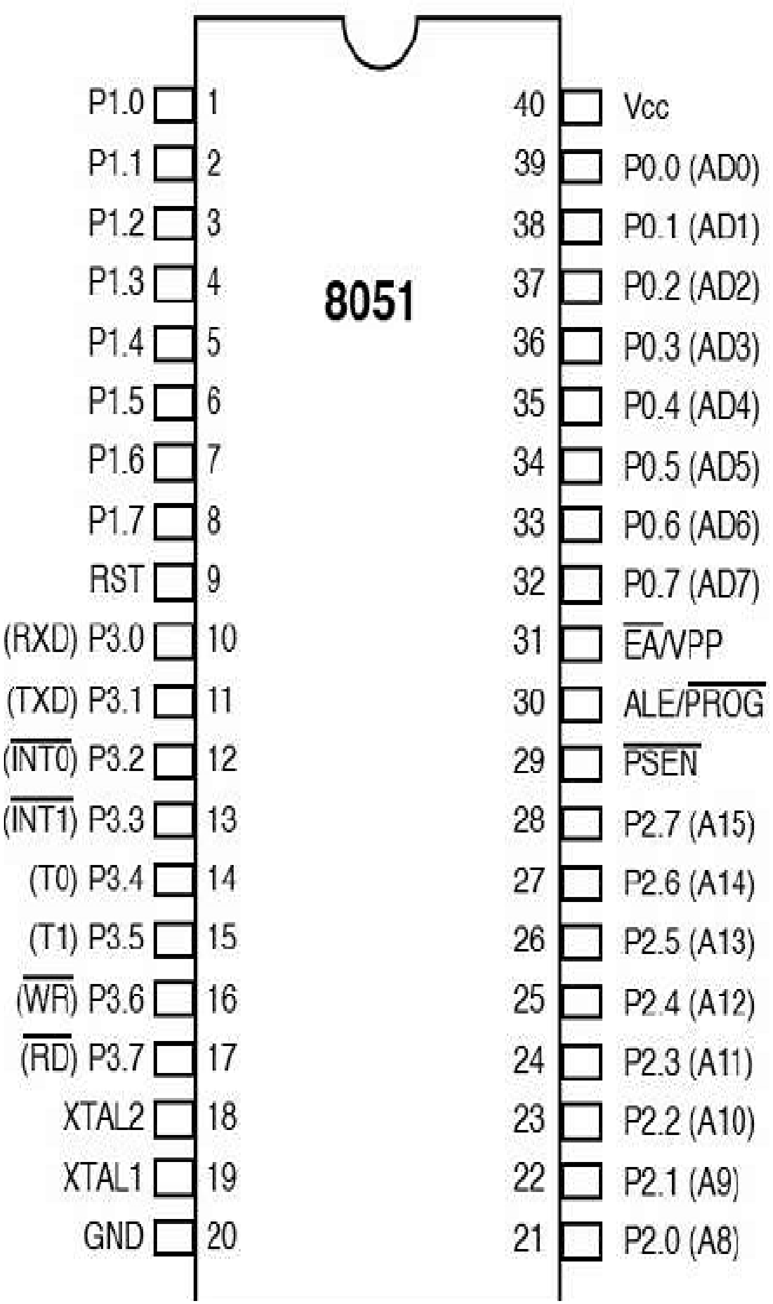
Pin 29 − This is PSEN pin which stands for Program Store Enable. It is used to read a signal from the external program memory.

Pin 31 − This is EA pin which stands for External Access input. It is used to enable/disable the external memory interfacing.

Pin 30 − This is ALE pin which stands for Address Latch Enable. It is used to demultiplex the address-data signal of port.

Pins 32 to 39 − These pins are known as Port 0. It serves as I/O port. Lower order address and data bus signals are multiplexed using this port.

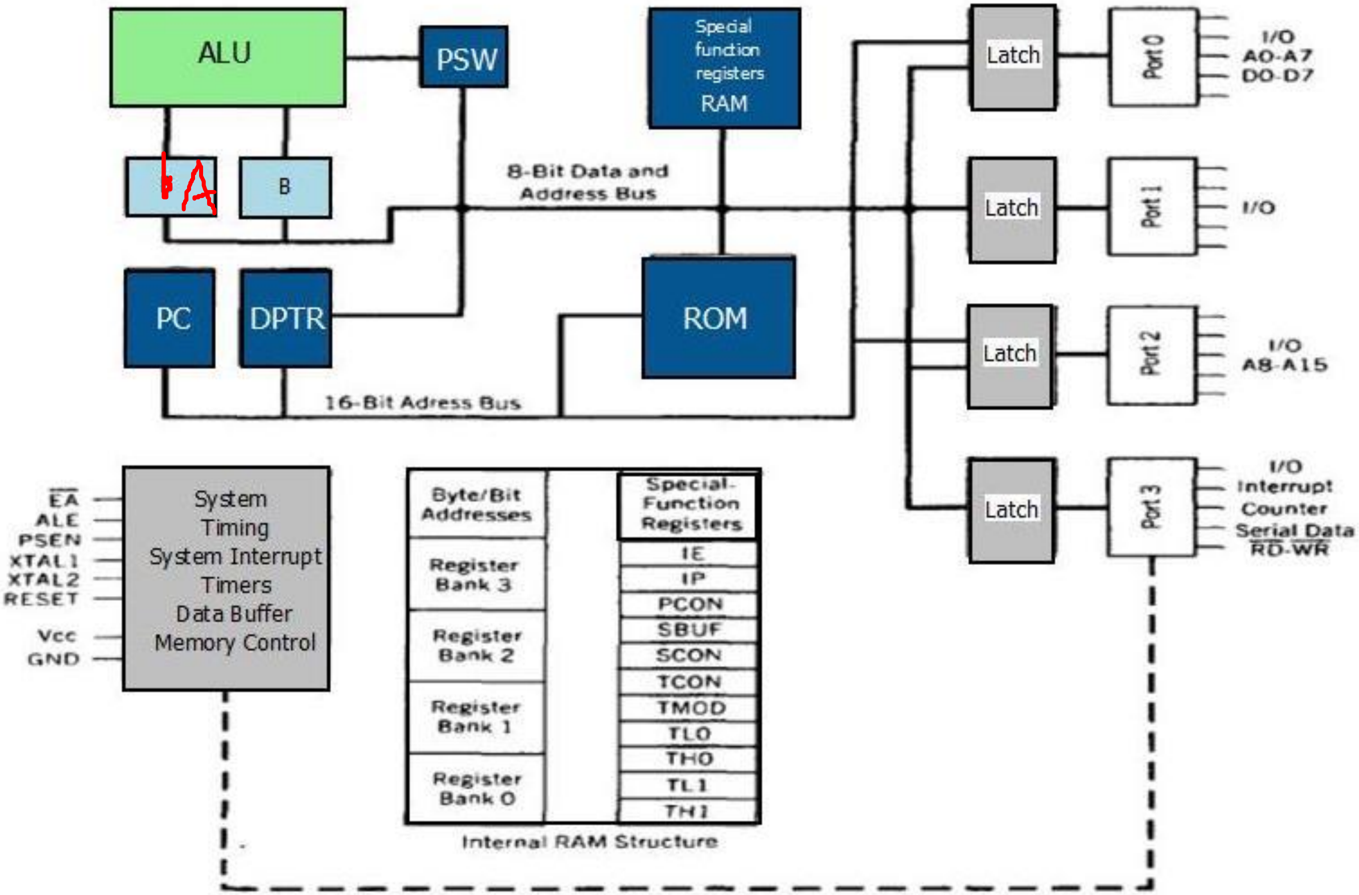
Pin 40 − This pin is used to provide power supply to the circuit.



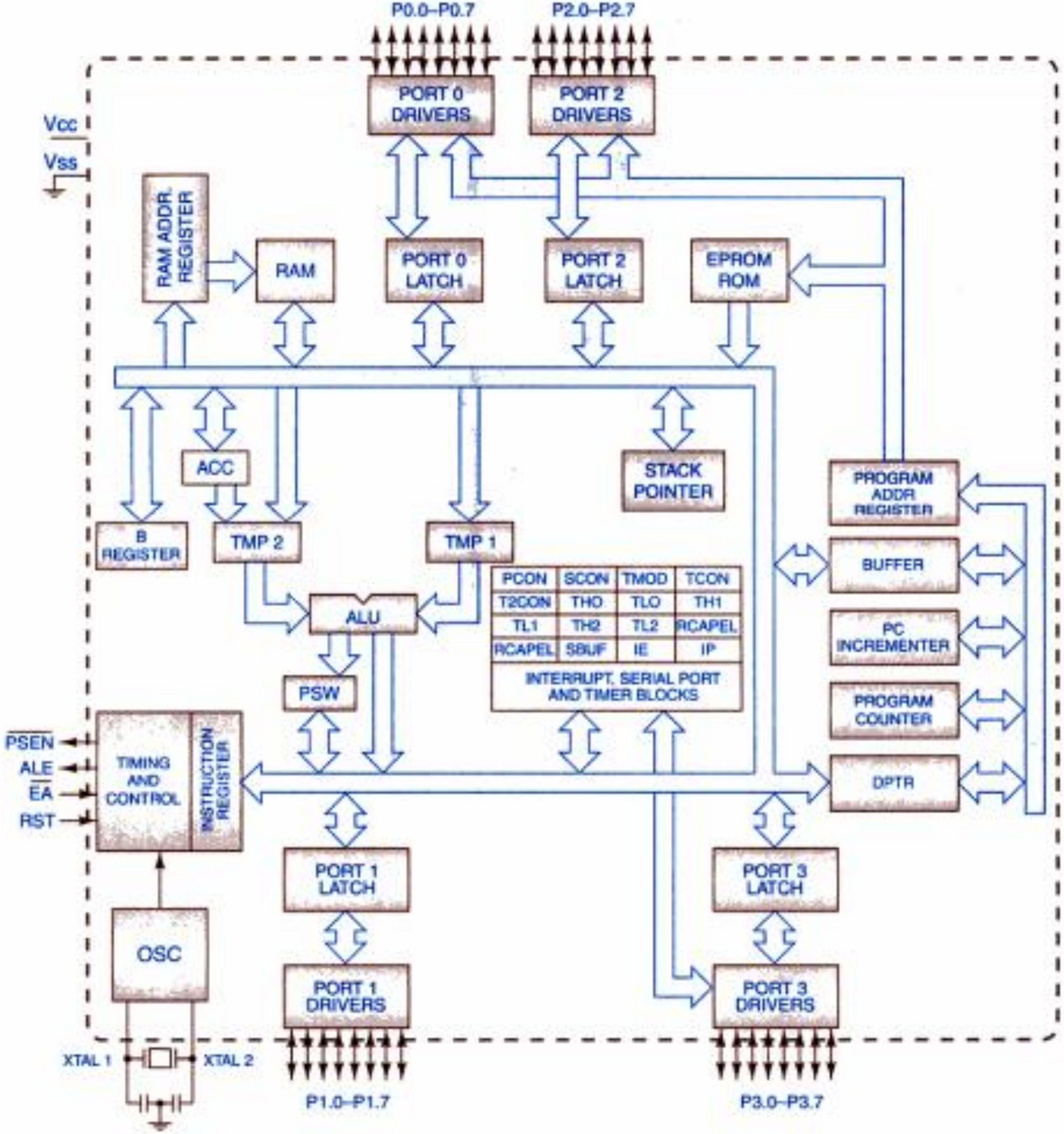
8051 Internal Architecture

The 8051 architecture include

- 8 bit CPU
- Memory
- Four 8 bit I/O
- Two 16 bit timers/counters
- Universal Asynchronous Receiver Transmitter(UART)



ALU
PSW
Special function registers RAM
A
B
8-Bit Data and Address Bus
PC
DPTR
ROM
16-Bit Adress Bus
Latch
Latch
Latch
Latch
Port 0
Port 1
Port 2
Port 3
I/O A0-A7 D0-D7
I/O
I/O A8-A15
I/O Interrupt Counter Serial Data RD WR
EA
ALE
PSEN
XTAL1
XTAL2
RESET
Vcc
GND
System Timing System Interrupt Timers Data Buffer Memory Control
Byte/Bit Addresses
Register Bank 3
Register Bank 2
Register Bank 1
Register Bank 0
Special Function Registers
IE
IP
PCON
SBUF
SCON
TCON
TMOD
TL0
TH0
TL1
TH1
Internal RAM Structure



PCON	SCON	TMOD	TCON
T2CON	THO	TLO	TH1
TL1	TH2	TL2	RCAPEL
RCAPEL	SBUF	IE	IP

INTERRUPT, SERIAL PORT AND TIMER BLOCKS

8051 Internal Architecture

PROCESSOR

The processor includes

- Arithmetic and Logic Unit
- Instruction Decoder and Timing Generation Unit
- Accumulator
- B register and status register

8051 Internal Architecture

ARITHMETIC AND LOGIC UNIT

- The accumulator is an 8 bit register
- In arithmetic and logic operations, one of the operands is in ‘A’ register

INSTRUCTION DECODER AND CONTROL

- Decodes the instructions and establish the sequence of events to flow

TIMING GENERATION UNIT

- Synchronizes all the microcontroller operations with the clock
- Generates control signals necessary for communication between the microcontroller and peripherals

8051 Internal Architecture

CPU REGISTERS

Accumulator (E0 H) register :-

- The accumulator is an 8 bit register
- In arithmetic and logic operations, one of the operands is in ‘A’ register
- After the arithmetic and logic operations, the result is stored in A register.

B (F0 H) register :-

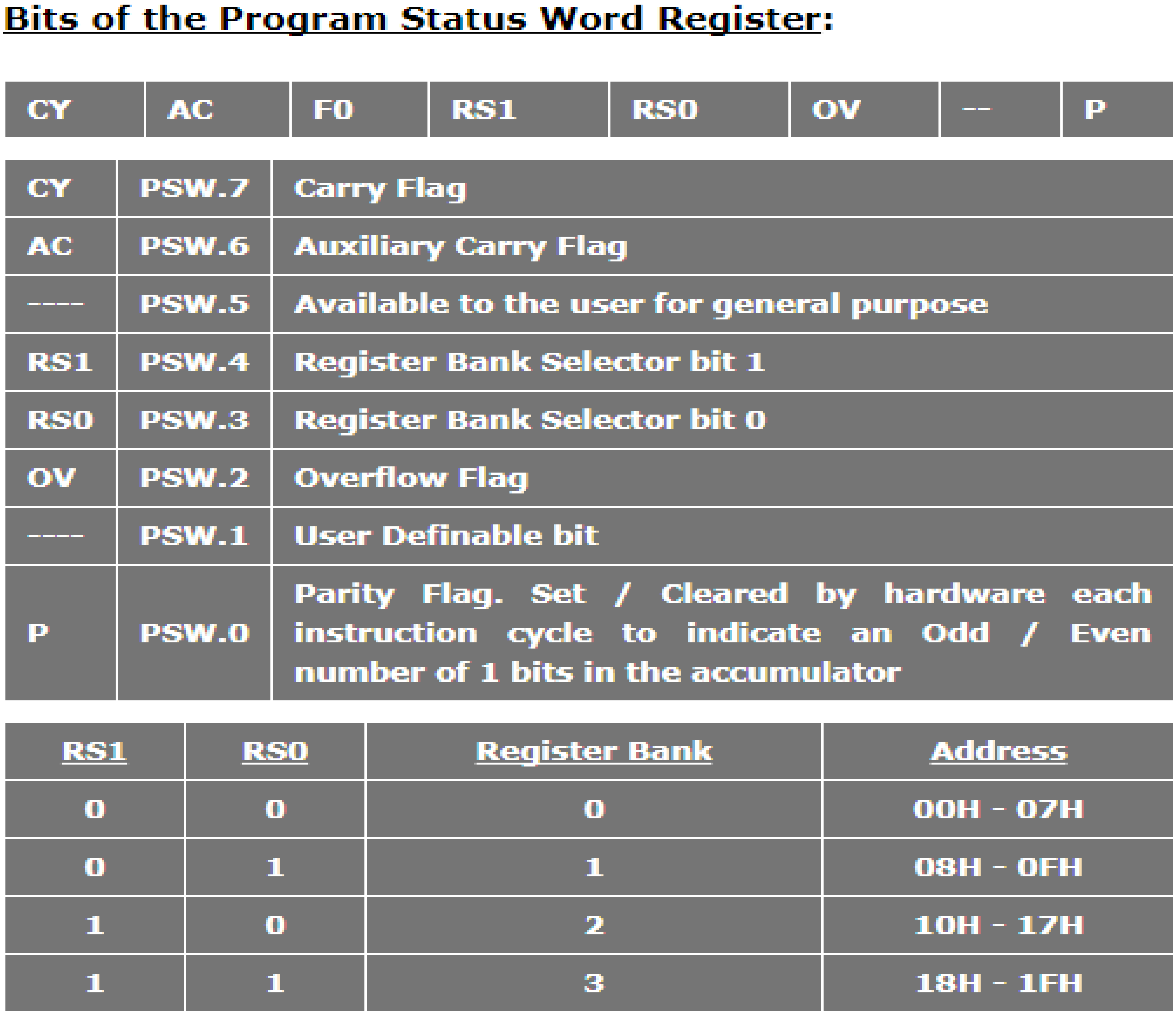
- 8 bit register used during multiply and divide operations
- In multiplication operation,the higher byte of the result is in ‘B’ register
- In division operation 8 bit divisor is in ‘B’ register and then remainder is stored in ‘B’ register

CPU Registers

Program Status Word (D0 H) (Flag Register)

- 8 bits wide, but only 6 bits of it are used
- remaining two unused bits are user-definable flags
- From the 6 bits, the 4 of them are *conditional flags*-
  - CY [carry]
  - AC [auxiliary carry]
  - P [Parity]
  - OV [overflow].
- other 2 bits are designated as RS0 and RS1, and are used to change the bank registers

FORMAT OF PSW



Bits of the Program Status Word Register:

CY	AC	F0	RS1	RS0	OV	--	P

CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag
----	PSW.5	Available to the user for general purpose
RS1	PSW.4	Register Bank Selector bit 1
RS0	PSW.3	Register Bank Selector bit 0
OV	PSW.2	Overflow Flag
----	PSW.1	User Definable bit
P	PSW.0	Parity Flag. Set / Cleared by hardware each instruction cycle to indicate an Odd / Even number of 1 bits in the accumulator

RS1	RS0	Register Bank	Address
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

8051 MEMORY ORGANIZATION

- divided into
  - ➢Program Memory and
  - ➢Data Memory

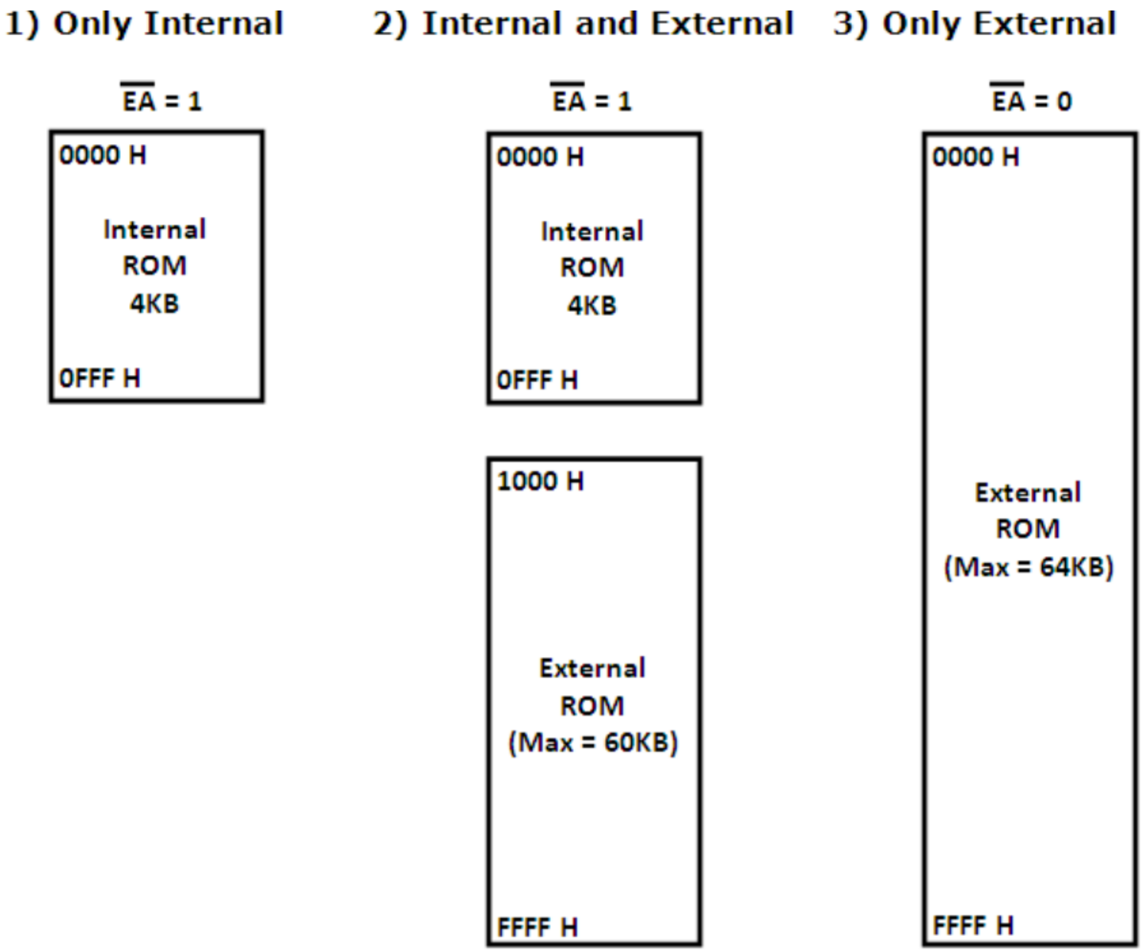
•Program Memory (ROM) is used for permanent saving program being executed

•Data Memory (RAM) is used for temporarily storing and keeping intermediate results and variables.

8051 PROGRAM MEMORY (ROM)

- Program Memory (ROM) is used for permanent saving program (CODE) being executed.
- 8051 memory organization allows external program memory to be added.
- **If EA=0** , the microcontroller completely ignores internal program memory and executes only the program stored in external memory.
- **If EA=1** In this case, the microcontroller executes first the program from built-in ROM, then the program stored in external memory.

8051 MEMORY ORGANIZATION



1) Only Internal

$\overline{EA} = 1$

0000 H

Internal ROM 4KB

0FFF H

2) Internal and External

$\overline{EA} = 1$

0000 H

Internal ROM 4KB

0FFF H

1000 H

External ROM (Max = 60KB)

FFFF H

3) Only External

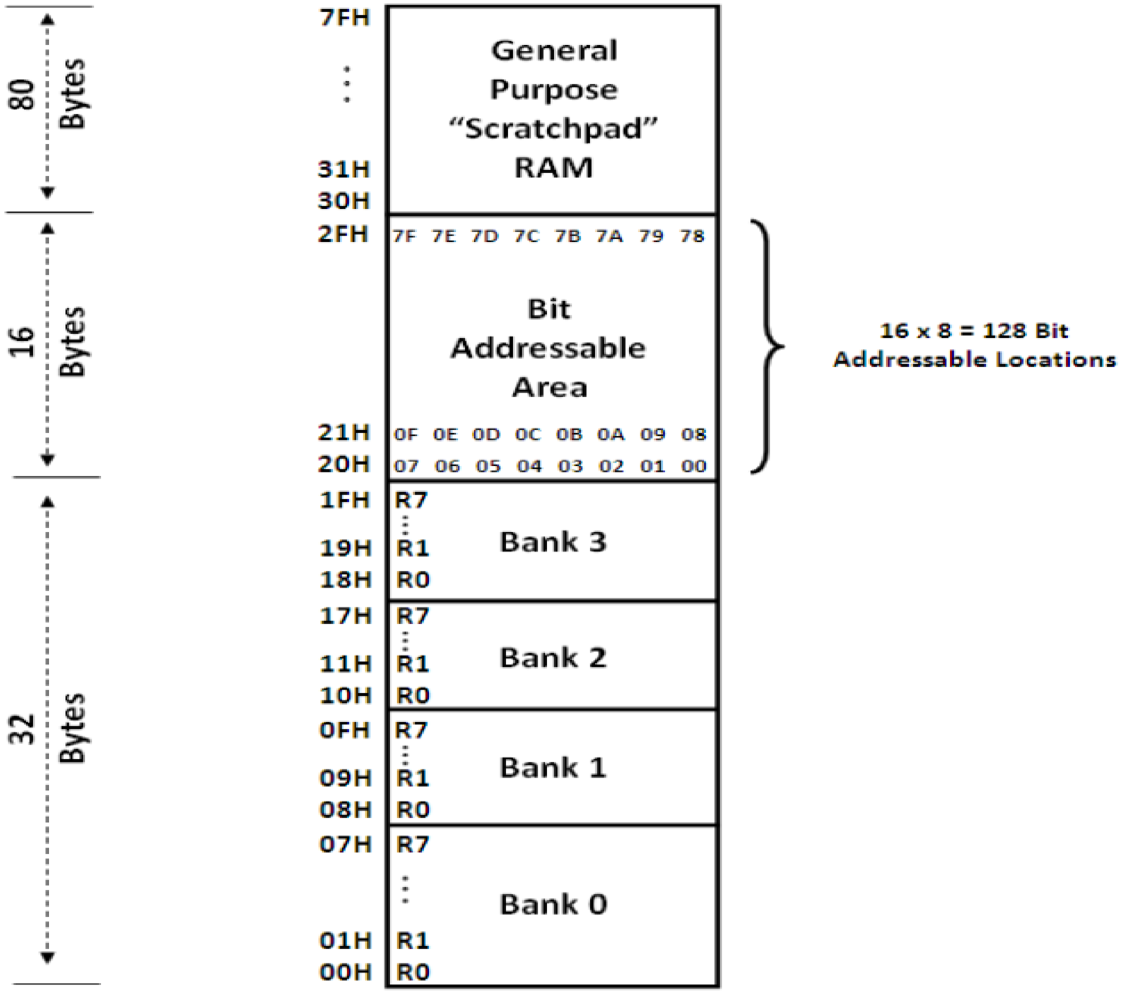
$\overline{EA} = 0$

0000 H

External ROM (Max = 64KB)

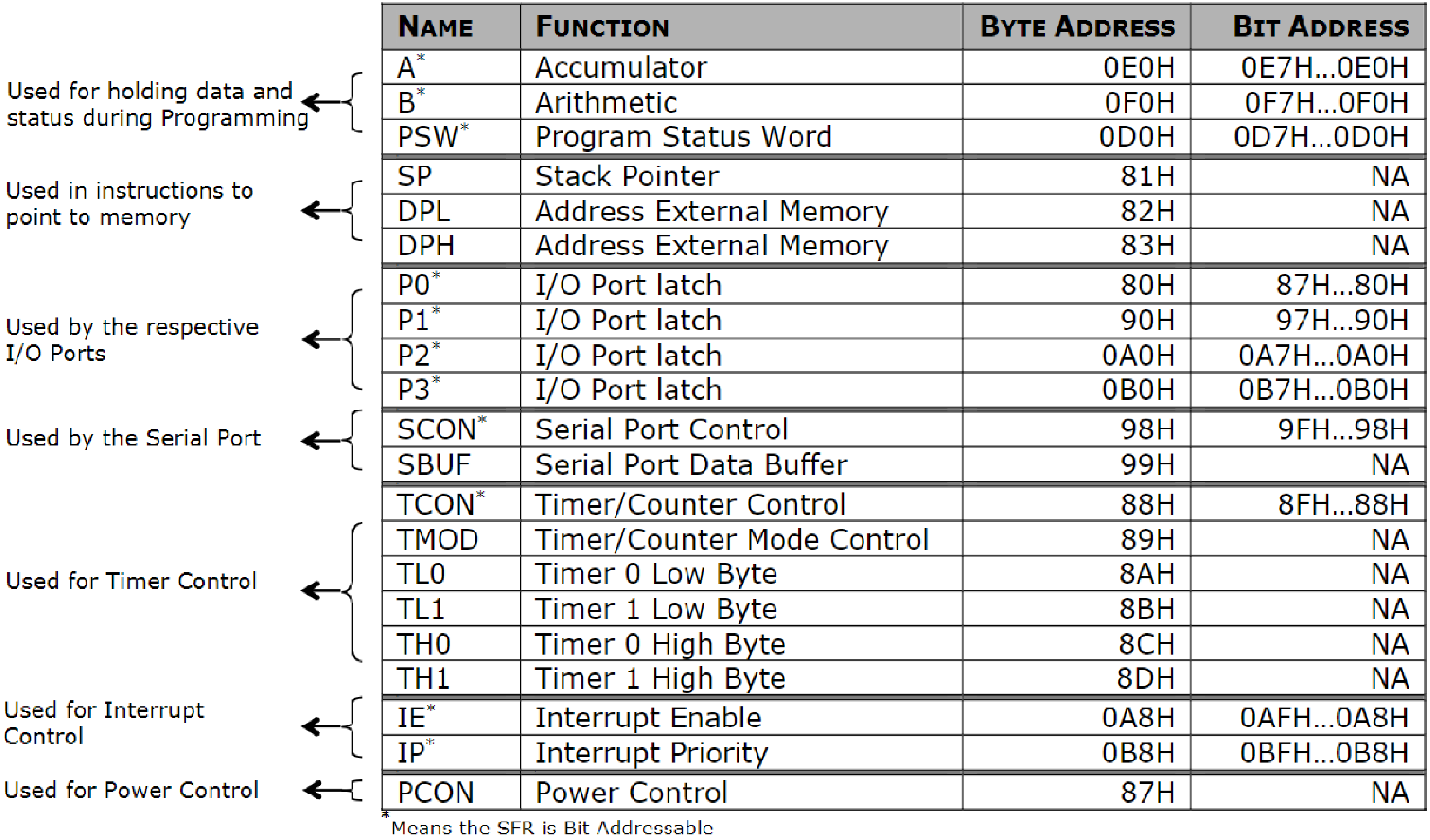
FFFF H

Structure of Internal RAM of 8051



Special Function Registers (SFRs of 8051)

- 21 8-bit SFR's are there in 8051



Used for holding data and status during Programming

Used in instructions to point to memory

Used by the respective I/O Ports

Used by the Serial Port

Used for Timer Control

Used for Interrupt Control

Used for Power Control

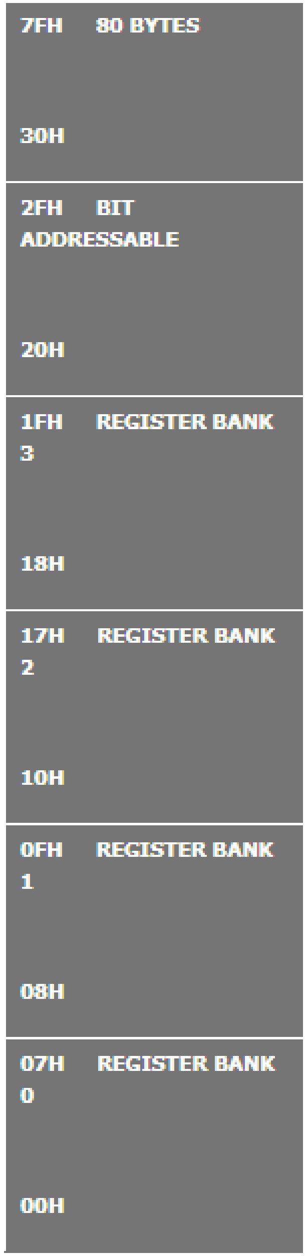
NAME	FUNCTION	BYTE ADDRESS	BIT ADDRESS
A*	Accumulator	0E0H	0E7H...0E0H
B*	Arithmetic	0F0H	0F7H...0F0H
PSW*	Program Status Word	0D0H	0D7H...0D0H
SP	Stack Pointer	81H	NA
DPL	Address External Memory	82H	NA
DPH	Address External Memory	83H	NA
P0*	I/O Port latch	80H	87H...80H
P1*	I/O Port latch	90H	97H...90H
P2*	I/O Port latch	0A0H	0A7H...0A0H
P3*	I/O Port latch	0B0H	0B7H...0B0H
SCON*	Serial Port Control	98H	9FH...98H
SBUF	Serial Port Data Buffer	99H	NA
TCON*	Timer/Counter Control	88H	8FH...88H
TMOD	Timer/Counter Mode Control	89H	NA
TL0	Timer 0 Low Byte	8AH	NA
TL1	Timer 1 Low Byte	8BH	NA
TH0	Timer 0 High Byte	8CH	NA
TH1	Timer 1 High Byte	8DH	NA
IE*	Interrupt Enable	0A8H	0AFH...0A8H
IP*	Interrupt Priority	0B8H	0BFH...0B8H
PCON	Power Control	87H	NA

*Means the SFR is Bit Addressable

8051 STACK AND REGISTER BANKS

- 128 bytes of RAM in the 8051
- assigned addresses 00 to 7FH.
- The 128 bytes are divided into three different groups as follows.
    - **32 bytes from locations 00 to 1F** set aside for register banks and the stack.
    - **16 bytes from locations 20H to 2FH** set aside for bit addressable Read/Write memory.
    - **80 bytes from locations 30H to 7FH** are used for read and write storage. These 80 bytes locations of RAM are widely used for the purpose of storing data and parameters by 8051 programmers.

RAM ALLOCATION IN 8051



7FH 80 BYTES

30H

2FH BIT ADDRESSABLE

20H

1FH REGISTER BANK 3

18H

17H REGISTER BANK 2

10H

0FH REGISTER BANK 1

08H

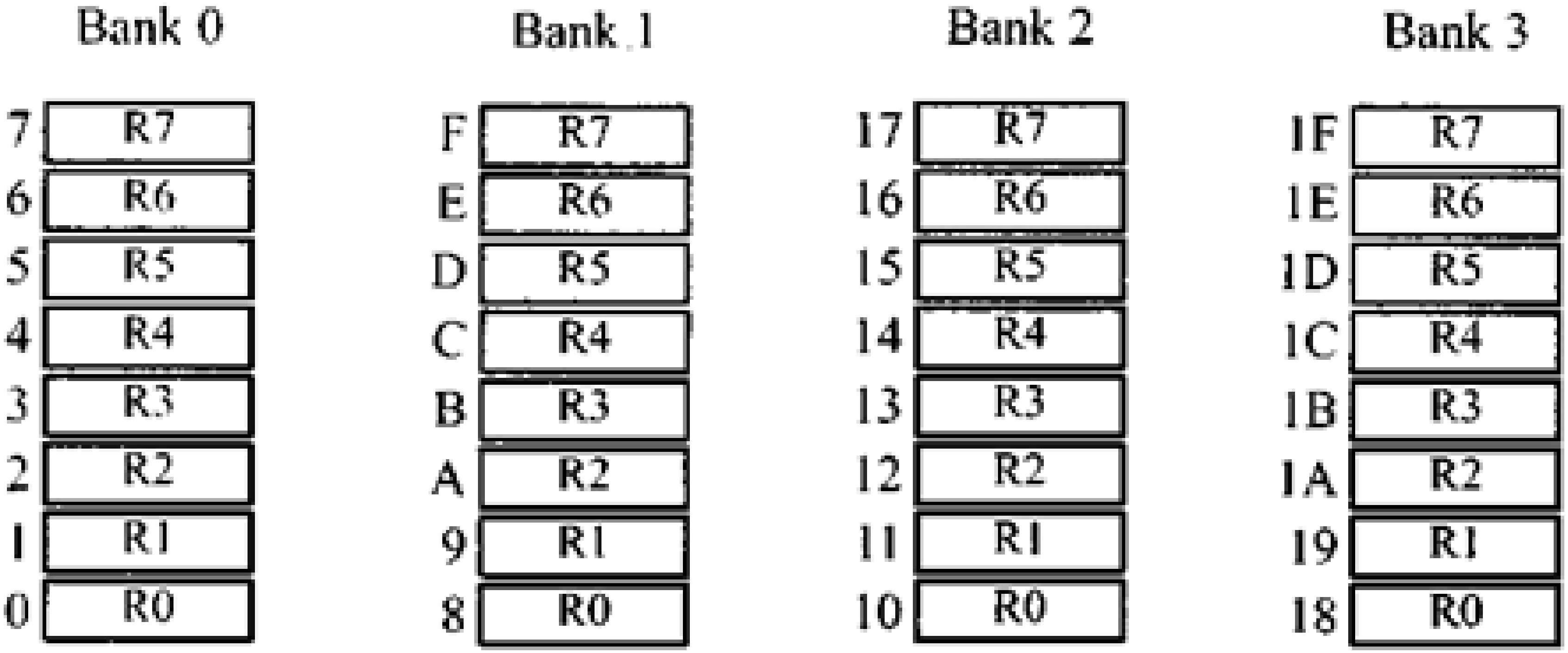
07H REGISTER BANK 0

00H

REGISTER BANKS IN THE 8051:

- 32 bytes of RAM are divided into 4 banks of registers in which each bank has 8 registers, R0 - R7.
- RAM locations from 0 to 7 are set aside for bank 0 of R0 - R7.
- Second bank of registers R0 - R7 starts at RAM location 08H and goes to location 0FH.
- Third bank of R0-R7 starts at memory location 10H and goes to location 17H.
- Fourth bank of R0-R7 starts at memory location 18H and goes to location 1FH.

REGISTER BANKS IN THE 8051:



Bank 0		Bank 1		Bank 2		Bank 3	
7	R7	F	R7	17	R7	1F	R7
6	R6	E	R6	16	R6	1E	R6
5	R5	D	R5	15	R5	1D	R5
4	R4	C	R4	14	R4	1C	R4
3	R3	B	R3	13	R3	1B	R3
2	R2	A	R2	12	R2	1A	R2
1	R1	9	R1	11	R1	19	R1
0	R0	8	R0	10	R0	18	R0

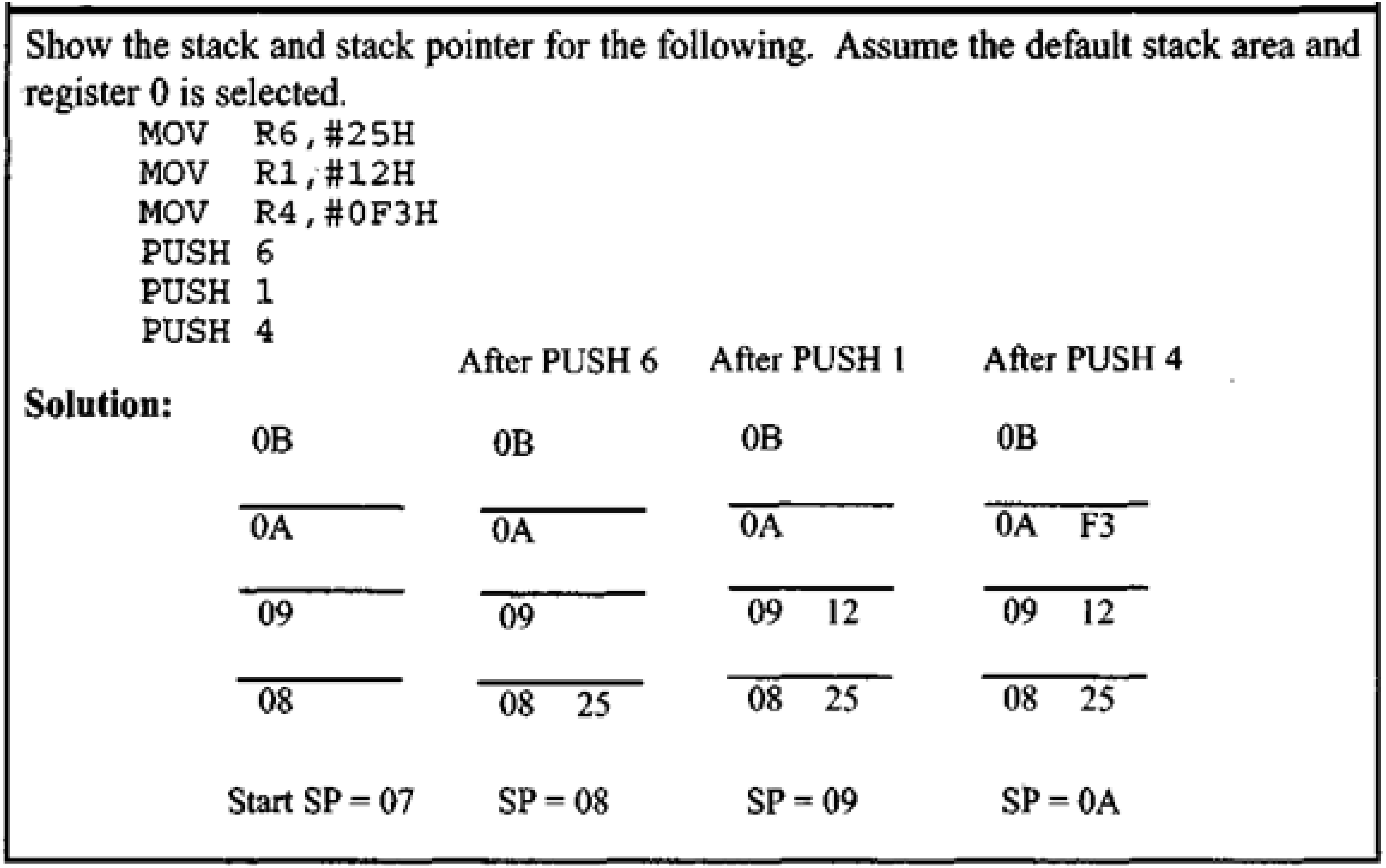
STACK IN THE 8051

- stack is a section of RAM used by the CPU to store information temporarily.
- this information could be data or an address.
- the register used to access the stack is called the SP (stack pointer) register.
- the stack pointer in the 8051 is only 8 bits wide, which means that it can take values of 00 to FFH.
- When the 8051 is powered up, the SP register contains value 07.
- This means that RAM location 08 is the first location used for the stack by the 8051.

PUSHING ONTO THE STACK

- the stack pointer (SP) points to the last used location of the stack.
- as data is pushed onto the stack, the stack pointer (SP) is incremented by one.
- as each PUSH is executed, the contents of the register are saved on the stack and SP is incremented by 1.
- to push the registers onto the stack their RAM addresses should be used.
- For example, the instruction “PUSH 1″ pushes register Rl onto the stack.

PUSHING ONTO THE STACK



Show the stack and stack pointer for the following. Assume the default stack area and register 0 is selected.

```
        MOV   R6,#25H
        MOV   R1,#12H
        MOV   R4,#0F3H
        PUSH  6
        PUSH  1
        PUSH  4
```

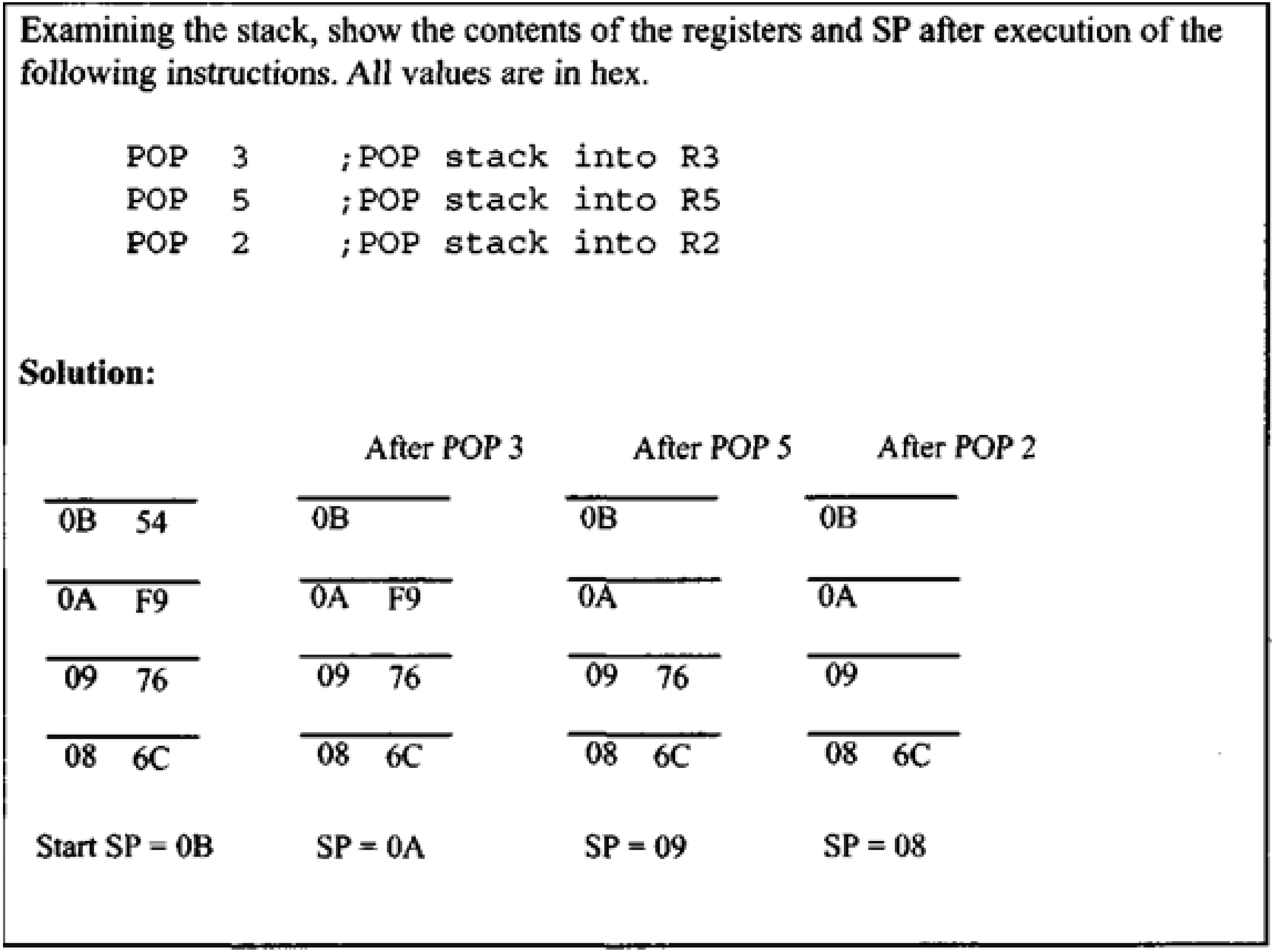
Solution:

	After PUSH 6	After PUSH 1	After PUSH 4
0B	0B	0B	0B
0A	0A	0A	0A F3
09	09	09 12	09 12
08	08 25	08 25	08 25
Start SP = 07	SP = 08	SP = 09	SP = 0A

POPPING FROM THE STACK

- Popping the contents of the stack back into a given register is the opposite process of pushing.
- With every pop, the top byte of the stack is copied to the register specified by the instruction and the stack pointer is decremented once

POPPING FROM THE STACK



Examining the stack, show the contents of the registers and SP after execution of the following instructions. All values are in hex.

```
POP  3     ;POP stack into R3
POP  5     ;POP stack into R5
POP  2     ;POP stack into R2
```

Solution:

		After POP 3		After POP 5		After POP 2	
0B	54	0B		0B		0B	
0A	F9	0A	F9	0A		0A	
09	76	09	76	09	76	09	
08	6C	08	6C	08	6C	08	6C
Start SP = 0B		SP = 0A		SP = 09		SP = 08	

8051 Interrupts

- **Interrupt Structure**: An interrupt is an external or internal event that disturbs the microcontroller to inform it that a device needs its service.
- The program which is associated with the interrupt is called the **interrupt service routine** (ISR) or **interrupt handler**.
- Upon receiving the interrupt signal the Microcontroller , finish current instruction and saves the PC on stack. Jumps to a fixed location in memory depending on type of interrupt Starts to execute the interrupt service routine until RETI (return from interrupt)
- Upon executing the RETI the microcontroller returns to the place where it was interrupted. Get pop PC from stack

8051 Interrupts

- The 8051 microcontroller has **FIVE interrupts in addition to Reset. They are**
  - ➢Timer 0 overflow Interrupt (TF0)
  - ➢Timer 1 overflow Interrupt (TF1)
  - ➢External Interrupt 0 (INT0)
  - ➢External Interrupt 1(INT1)
  - ➢Serial Port events (buffer full, buffer empty, etc) Interrupt (R1=T1)

8051 Interrupts

- Each interrupt has a specific place in code memory where program execution (interrupt service routine) begins.
  - ➢External Interrupt 0: 0003 H
  - ➢Timer 0 overflow: 000B H
  - ➢External Interrupt 1: 0013 H
  - ➢Timer 1 overflow: 001B H
  - ➢Serial Interrupt: 0023 H

(Interrupt lists above in the decreasing order of priority)

MODULE-5

Part 2

8051 Addressing Modes

- An **addressing mode** refers to how you are addressing a given memory location. The five different ways of addressing are as follows −
- Immediate addressing mode
- Direct addressing mode
- Register direct addressing mode
- Register indirect addressing mode
- Indexed addressing mode

Immediate Addressing Mode

- Data operand is a constant and it is a part of the instruction itself
- The data must be preceded by a # sign.
- Eg1:-MOV A, #63H

The data (constant) 63 is moved to the accumulator register

- Eg2: MOV DPTR, # 2000H

Direct addressing mode

- Only for Internal RAM and the SFRs

Eg:-**MOV R1, 42H** : Move the contents of RAM location 42 into R1 register

MOV 49H,A : Move the contents of the accumulator into the RAM location 49

- MOV 30H,#30H
- MOV 01H,00H --------→allowed in MC (not in MP)
- MOV A,80H -------------→ A← P0 {SFR}
- MOV 30H,A

Register addressing mode

- Data operand to be manipulated lies in one of the registers
- Eg:-
  - **MOV A,R0** : Move the contents of the register R0 to the accumulator
  - **ADD A,R6** :Add the contents of R6 register to the accumulator
  - **MOV P1, R2 :** Move the contents of the R2 register into port 1

Note: MOV R0,R1 (not allowed)

So use MOV A, R1

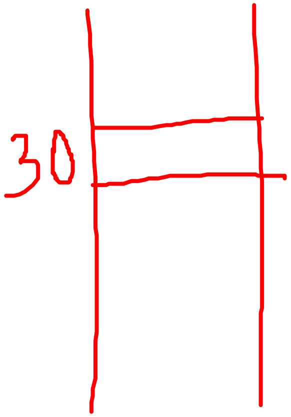
MOV R0,A

Register Indirect addressing mode

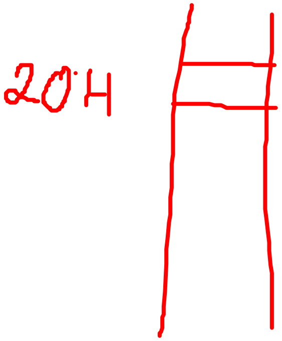
- This is for internal RAM & External RAM
- To access Int RAM ,8-bit addresses required(R0/R1 only)
- To access ext RAM , 16-bit addresses required(DPTR)
- Eg:- Internal RAM
  - ➢ **MOV A,@ R0 :**Move the contents of RAM location whose address is in R0 into **A (accumulator)**
  - ➢ **MOV @ R1 , B** : Move the contents of B into RAM location whose address is held by R1

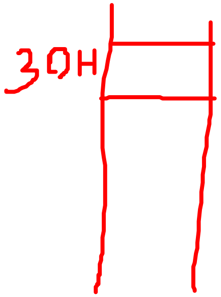
Eg:- External RAM

```
MOVX  A,@DPTR
MOVX  DPTR, @A
```



```
mov A,30H
      OR
mov R0,#30H
mov A,@R0
```





```
MOV R0,#20H
MOV R1,#30H
MOV A,@R0
mov @R1,A
 INC R0
 INC R1
```

Indexed addressing mode

- Only for ROM
- Address obtained from two registers
- Eg:-
  - ➢ MOVC A, @ A+DPTR
  - ➢   MOVC A, @A+PC

8051 INSTRUCTION SET

- **1. Instruction Timings**
- **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.
- **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.

The instructions of 8051

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branch instructions
5. Subroutine instructions
6. Boolean/Bit manipulation instructions

DATA TRANSFER INSTRUCTIONS

a. Move the contents of a register Rn to A

i. MOV A,R2

b. Move the contents of a register A to Rn

i. MOV R4,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

- **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 20H, 30H
  - 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

- **i. 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

- **j. 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

- k.PUSH 25H

  SP🡠 SP+1 ,

Then [SP]🡠 [25H]

- l.POP 25H

First [25H] 🡠[SP] ,Then SP 🡠 SP-1

ARITHMETIC INSTRUCTIONS

- **1.Addition**

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, @R0

iv ADDC A, @R1

1 2 F F +   CF = 1

0 0 0 1

1 3 0 0

B=1

$$\begin{array}{r} 47\ - \\ 19 \\ \hline 28 \end{array}$$

2.Subtraction

- **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
- Simple Subtraction also use SBB, so use CLR C instn before subtraction

3.Multiplication

- **MUL AB.**

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

4.Division

- **DIV AB.**

Eg:

- MOV A,#0F5H ;
- MOV B,#45H ;
- DIV AB ;

A ←----Quotient, B←----Remainder

6. Increment:

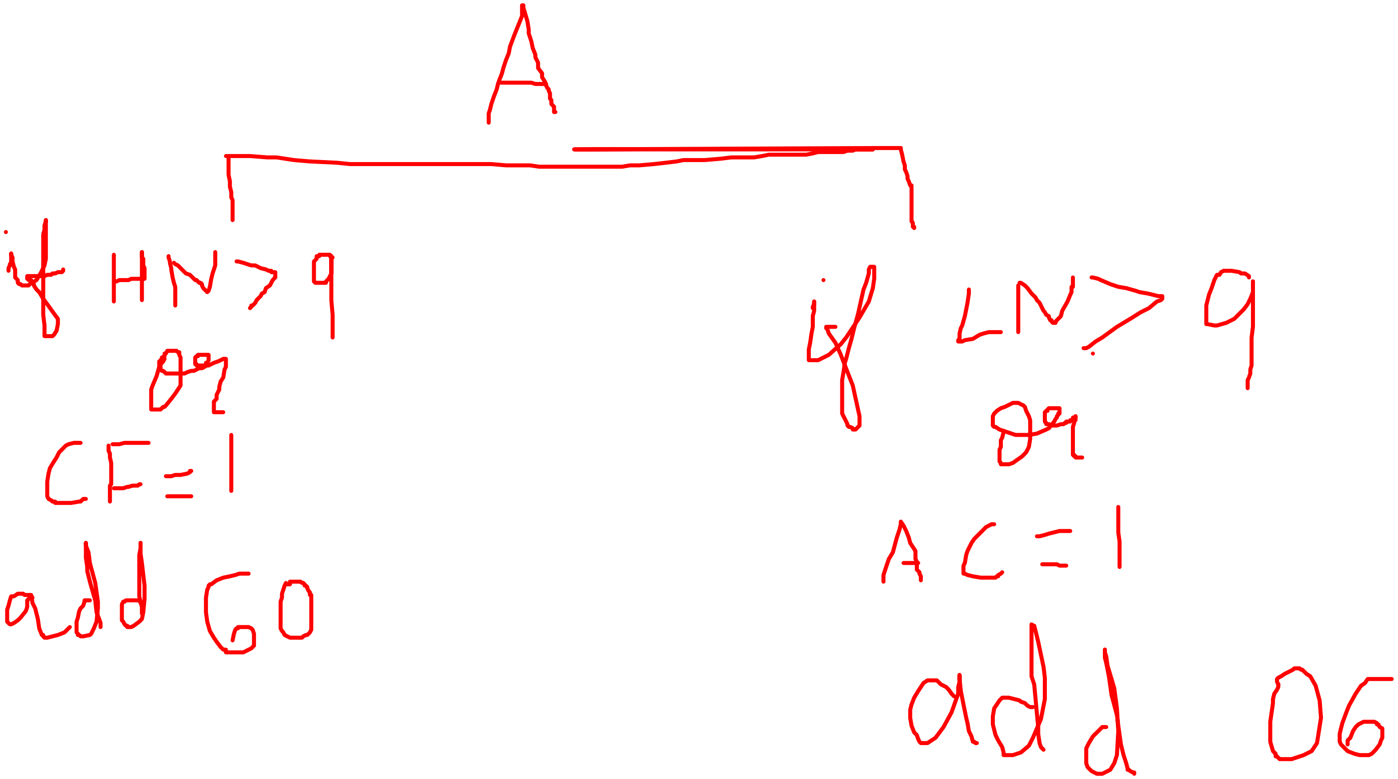
- INC increments the value of source by 1. If the initial value of register is FFh, incrementing the value will cause it to reset to 0. The Carry Flag is not set when the value "rolls over" from 255 to 0. In the case of "INC DPTR", the value two-byte unsigned integer value of DPTR is incremented. If the initial value of DPTR is FFFFh, incrementing the value will cause it to reset to 0.
- INC A
- INC R1
- INC 25H
- INC @R0
- INC DPTR

7.Decrement:

- 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 FFh to 0
- DEC A
- DEC R0
- DEC 25H
- DEC @R0
- DEC DPTR (Invalid)

5. **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.



A

if HN > 9
or
CF = 1
add 60

if LN > 9
or
AC = 1
add 06

LOGICAL INSTRUCTIONS

1. **Logical AND**

- **ANL** destination, source
- Eg:-
- ANL A,#25 ANL A,@R0 ANL 25H,A
- ANL A, R0 ANL A,20H ANL 25H, #25
- Note:

Clear LN, Suppose A= 0011 0101 (AND LN with 0’s & HN with 1’s)

ANL A, #0F0H

Logical OR

2.ORL destination, source

- Eg:-
- ORL A,#25 ORL A,@R0 ORL 25H,A
- ORL A, R0 ORL A,20H ORL 25H, #25
- Note:

SET LN , Suppose A= 0011 0101 (OR LN with 1's & HN with 0's)

ORL A, #0FH

Logical Ex-OR

- **3.XRL destination, source**
- XRL A,#25 XRL A,@R0 XRL 25H,A
- XRL A, R0 XRL A,20H XRL 25H, #25
- Note:

Complement LN, Suppose A= 0011 0101 (XRL LN with 1’s & HN with 0’s)

XRL A, 0FH

4.Logical NOT

CPL A, CPL C, CPL bit address

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

Rotate Instructions

- **RR A**
- This instruction is rotate right the accumulator. 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- Unconditional

- Branch instruction -2 types JUMP & CALL

There are 3 types of jump instructions. They are:-

- 1. Short Jump (SJMP radd)
- 2. Absolute Jump (AJMP sadd)
- 3. Long Jump(LJMP ladd)

CALL-

- LCALL sub
- SCALL sub

RETURN instn – 2 types

- RET
- RETI

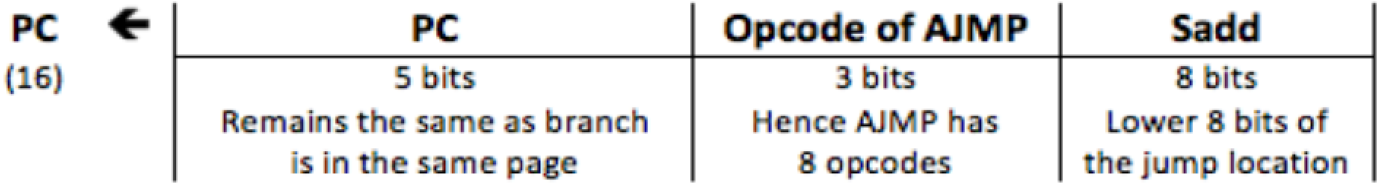
- .1.Short Jump (SJMP radd)
    - Radd- 8-bit signed no
    - Ranges (-128 to +127)
    - Radd is calculated as the relative distance from the next instn to the branch locn
    - Telling how far we jump not where we want to jmp
    - Eg: SJMP 08H
- 2. Absolute Jump (AJMP sadd)
- Here entire 64 KB divided into 32 pages, each page is of 2 KB
- As JMP is in the same page ,only 11 bits of address will change

Syntax: **AJMP sadd**;// Absolute Jump using the short address

Range: **max 2KB** as long as the Jump is within the **Same Page**

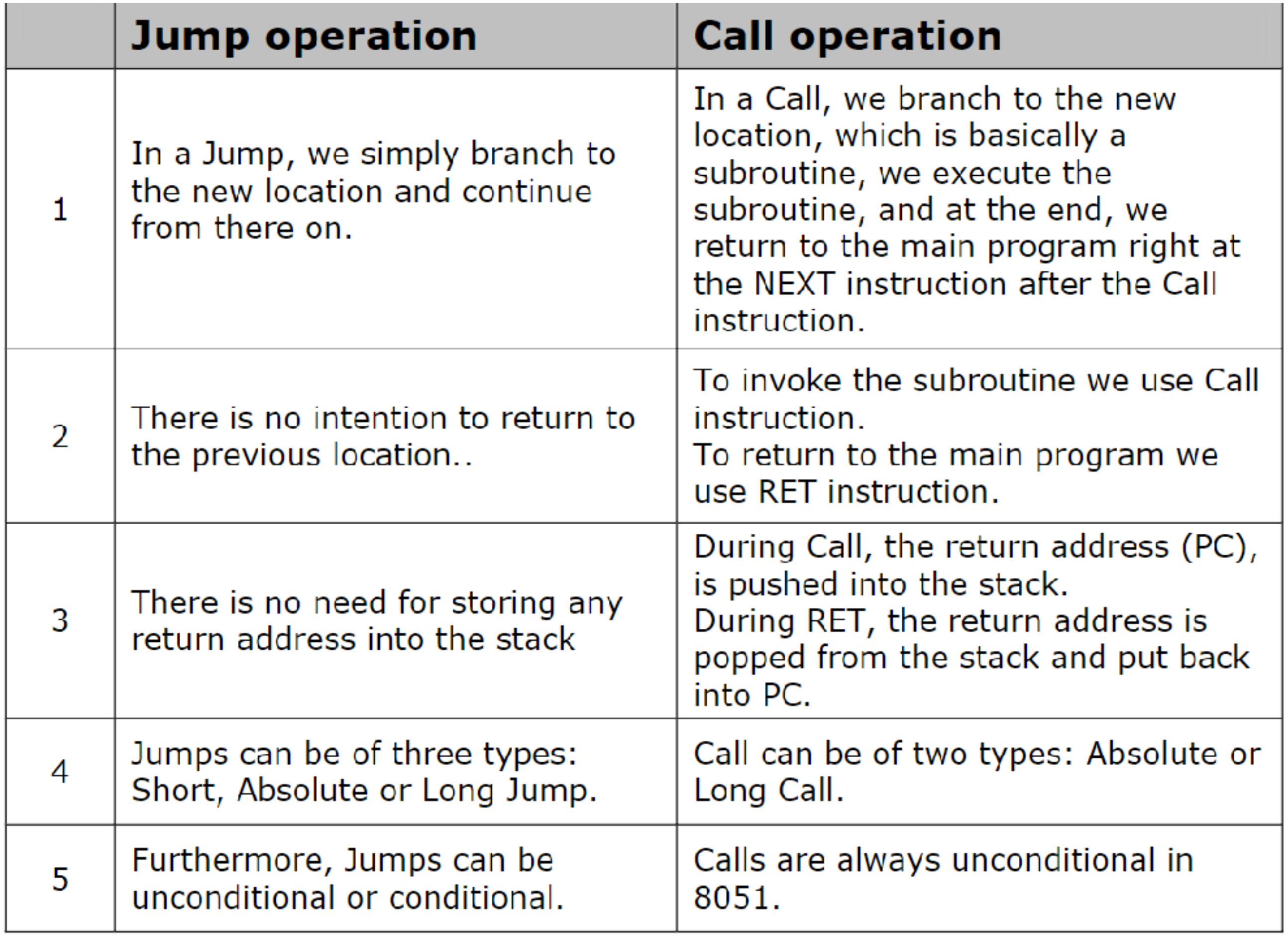
Size of instruction: **2 Bytes** (Opcode of AJMP= 1Byte, sadd = 1Byte)

New address calculation:



PC ←	PC	Opcode of AJMP	Sadd
(16)	5 bits Remains the same as branch is in the same page	3 bits Hence AJMP has 8 opcodes	8 bits Lower 8 bits of the jump location

- 3. Long Jump(LJMP ladd)
  - ladd- 16 bit address (0000H to FFFFH)
  - 3 bytes (1 byte opcode +2 byte ladd)
  - Eg: LJMP 5000H
  - LCALL also like this



	Jump operation	Call operation
1	In a Jump, we simply branch to the new location and continue from there on.	In a Call, we branch to the new location, which is basically a subroutine, we execute the subroutine, and at the end, we return to the main program right at the NEXT instruction after the Call instruction.
2	There is no intention to return to the previous location..	To invoke the subroutine we use Call instruction. To return to the main program we use RET instruction.
3	There is no need for storing any return address into the stack	During Call, the return address (PC), is pushed into the stack. During RET, the return address is popped from the stack and put back into PC.
4	Jumps can be of three types: Short, Absolute or Long Jump.	Call can be of two types: Absolute or Long Call.
5	Furthermore, Jumps can be unconditional or conditional.	Calls are always unconditional in 8051.

	RET	**RETI**
1	RET is used at the end of an ordinary Subroutine	RETI is used at the end of an ISR
2	RET will simply return back to the next instruction of the main program.	RETI will not only return back to the next instruction of the main program, but will also re-enable the interrupts, by making EA bit ← 1.

3	Operation:
	POP PC; PCH← [SP]
	PCL← [SP-1]
	SP← SP-2
		Operation:
		POP PC; PCH← [SP]
		PCL← [SP-1]
		SP← SP-2
		EA ←1; Enables interrupts by making EA bit "1" in the IE-SFR.

Conditional jump instructions

- 1.CJNE A,#25H ,label

(Compare & jmp if not equal)

- CJNE A,25H ,label
- CJNE R0,#25H ,label
- CJNE @R0,25H ,label
- 2. DJNZ count, label (Decrement and jmp if not Zero)
- DJNZ R2,BACK
- DJNZ 25H,BACK

- JC label
- JNC label
- JZ label
- JNZ label

Bit wise Conditional jump instructions

- JB bit,label

Eg: JB P0.2,down

- JNB bit,label

Eg: wait :JNB TF0,wait

- JBC bit,label *(jmp if bit =1 then goto label ,while jumping clear bit also)*

Eg: JBC P0.7,down, *(if p0.7=1 jmp to locn down and make po.7=0)*

CALL /SUBROUTINE INSTRUCTIONS

- Subroutines are handled by CALL and RET instructions There are two types of CALL instructions
- **LCALL address(16 bit)**
- This is long call instruction which unconditionally calls the subroutine located at the indicated 16 bit address. This is a 3 byte instruction. The LCALL instruction works as follows.
-
- a. During execution of LCALL, [PC] = [PC]+3; (if address where LCALL resides is say, 0x3254; during execution of this instruction [PC] = 3254h + 3h = 3257h

  b. [SP]=[SP]+1;

  c. [[SP]] = [PC7-0];

  d. [SP]=[SP]+1;

  e. [[SP]] = [PC15-8];

ACALL address(11 bit)

- This is absolute call instruction which unconditionally calls the subroutine located at the indicated 11 bit address. This is a 2 byte instruction. The ACALL instruction works as follows.

a. During execution of ACALL, [PC] = [PC]+2; (if address where LCALL resides is say, 0x8549; during execution of this instruction [PC] = 8549h + 2h = 854Bh

b. [SP]=[SP]+1

c. [[SP]] = [PC7-0]

d. [SP]=[SP]+1; (SP increments again and [SP]=09)

e. [[SP]] = [PC15-8]

With these the address (0x854B) which was in PC is stored in stack.

f. [PC10-0]= address (11 bit); the new address of subroutine is loaded to PC. No flags are affected.

RET instruction

RET instruction pops top two contents from the stack and load it to PC.

- g. [PC15-8] = [[SP]]
- h. [SP]=[SP]-1
- i. [PC7-0] = [[SP]]
- j. [SP]=[SP]-1; (SP decrements again)

Boolean/ 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

    Eg: ANL C, P0.2

    ANL C, 25H
  - b. ANL C, /BIT;

    Eg: ANL C, / P0.3 (C ^/P0.3)
- **2. LOGICAL OR**
  - a. ORL C,BIT Eg: ORL C, P0.2
  - b. ORL C, /BIT; Eg: ORL C, /P0.2

3. CLR bit

- a. CLR bit   Eg: CLR P0.2
- b. CLR C   (C←-- 0)

4. CPL bit

a. CPL bit   Eg: CPL P0.2

b. CPL C

c. CPL 02H

5. SETB C

6. SETB bit   eg: SETB PSW.0

7. CLR C

8. CLR bit eg: CLR P0.1

9. MOV C,bit

Eg: MOV C, P0.3
      MOV C,25H

10. MOV bit,C

Addition of two 8-bit nos

```
MOV  DPTR, #4200H
MOVX  A, @DPTR
MOV  B, A
INC  DPTR
MOVX  A, @DPTR
MOV  R0, 00H
ADD A, B
JNC  L1
INC RO
```

```
L1 :  INC DPTR
        MOVX @DPTR, A
         INC DPTR
          MOV A, R0
          MOVX @DPTR, A
HERE : SJMP   HERE
```

4203	Carry
4202	50
4201	30
4200	20

$A \leftarrow A+B$

$B = 20$

$R = 00$

$A = 30$

Subtraction of two 8-bit nos

```
MOV  DPTR, #4200
MOVX  A, @DPTR
MOV  B, A
INC  DPTR
MOVX  A, @DPTR
MOV  R0, 00H
CLR C
SUBB A, B
JNC  L1
CPL A
INC A
INC Ro
```

```
L1 :  INC DPTR
        MOVX @DPTR, A
         INC DPTR
          MOV A, Ro
          MOVX @DPTR, A
HERE : SJMP   HERE
```

Multiply two 8-bit nos

```
MOV DPTR, #4200H
MOVX A, @DPTR
MOV B,A
INC  DPTR
MOVX A, @DPTR
MUL AB
INC DPTR
MOVX @DPTR, A
MOV A, B
INC DPTR
MOVX @DPTR, A
HERE : SJMP   HERE
```

- [4200] =FF
- [4201]=FF

FF x
FF

FE 01

DIVIDE TWO 8-BIT NOS

```
MOV DPTR, #4200H
MOVX A, @DPTR
MOV Ro , A
INC  DPTR
MOVX A, @DPTR
MOV B, A
MOV A,Ro
DIV  AB
INC DPTR
MOVX @DPTR, A
MOV A, B
INC DPTR
MOVX @DPTR, A
HERE : SJMP   HERE
```

Find the largest no in the array

```
MOV DPTR ,#5000H
MOVX A, @DPTR
MOV Ro, 05H
L1 : MOV B, A
L3:DJNZ R0,L2
    SJMP L4
L2:  INC DPTR
       MOVX A, @DPTR
       CJNE A,B ,NEG
        SJMP L3
 NEG:    JC L3          → A < B
       SJMP L1
```

```
L4:  MOV A, B
        INC DPTR
         MOVX @DPTR,A
HERE : SJMP   HERE
```

WAP to add the contents of Internal RAM locations 40H and 41H. Store Result at 42H and Carry at 43H

```
         MOV 43H, #00H          ; Initialize Carry as "0"
         MOV A, 40H             ; Read first number
         ADD A, 41H             ; Add second number
         JNC SKIP               ; If no Carry, directly store the Sum
         INC 43H                ; Store Carry as "1"
 SKIP:   MOV 42H, A             ; Store Sum
 HERE:   SJMP HERE              ; End of program
```

Q2 WAP to multiply the numbers B2H and 2FH. Store Result in registers R0 (LSB) and R1 (MSB) of Bank 2.

```
SOLN:  MOV A, #0B2H          ; Read first number
       MOV 0F0H, #2FH        ; Read second number
       MUL AB                ; Multiply the operands
       SETB PSW.4
                             ; Select Bank 2
       CLR  PSW.3
       MOV R0, A             ; Store LSB of result
       MOV R1, 0F0H          ; Store MSB of result
HERE:  SJMP HERE             ; End of program
```

Q3 WAP to add a series of 10 numbers. The series begins from location 20H in Internal RAM. Store the result at locations 30 and 31H.

```
   SOLN:  MOV R0, #20H        ; Initialize Source address
          MOV R1, #0AH        ; Initialize count of 10
          CLR A               ; A register will accumulate the Sum
          MOV 0F0H, #00H      ; B register will accumulate the Carry
 REPEAT:  ADD A, @R0          ; Add the current element
          JNC SKIP            ; If no carry, then directly proceed ahead
          INC 0F0H            ; If there is a carry, increment B Register
   SKIP:  INC R0              ; Increment source address
          DJNZ R1, REPEAT     ; Decrement count. If Count is NOT ZERO then repeat.
          MOV 30H, A          ; Store Sum
          MOV 31H, 0F0H       ; Store Carry
   HERE:  SJMP HERE           ; End of program
```

Q4

WAP to add a series of 10 "BCD" numbers. The series begins from location 20H. Store the result at locations 30 and 31H.

```
 SOLN:   MOV R0, #20H        ; Initialize Source address
         MOV R1, #0AH        ; Initialize count of 10
         CLR A               ; A register will accumulate the Sum
         MOV 0F0H, #00H      ; B register will accumulate the Carry
REPEAT:  ADD A, @R0          ; Add the current element
         DA A                ; Decimal adjust as the operands were BCD numbers
         JNC SKIP            ; If no carry, then directly proceed ahead
         INC OFOH            ; If there is a carry, increment B Register
 SKIP:   INC R0              ; Increment source address
         DJNZ R1, REPEAT     ; Decrement count. If Count is NOT ZERO then repeat.
         MOV 30H, A          ; Store Sum
         MOV 31H, 0F0H       ; Store Carry
 HERE:   SJMP HERE           ; End of program
```

Q5 **WAP to add a series of 10 numbers. The series begins from location 2000H in External RAM. Store the result at locations 3000 and 3001H.**

```
  SOLN:  MOV DPTR, #2000H     ; Initialize Source address
         MOV R1, #0AH         ; Initialize count of 10
         CLR A                ; “A”  will accumulate the Sum, and ALSO get the data from Ext. RAM
         MOV 0F0H, #00H       ; B register will accumulate the Carry
         MOV R0, #00H         ; R0 will be used to
REPEAT:  MOVX A, @DPTR        ; Get the data from Ext RAM
         ADD A, R0            ; Add the data to the previous sum
         JNC SKIP             ; If no carry, then directly proceed ahead
         INC OFOH             ; If there is a carry, increment B Register
  SKIP:  INC DPTR             ; Increment source address
         MOV R0, A            ; Store current Sum in R0  from next iteration
         DJNZ R1, REPEAT      ; Decrement count. If Count is NOT ZERO then repeat.
         MOV DPTR, #3000H     ; DPTR gets address to store the Sum
         MOVX @DPTR, A        ; Store Sum
         INC DPTR             ; DPTR gets address to store the Carry
         MOV A, OFOH          ; Transfer Carry from B to A register
         MOVX @DPTR, A        ; Store Carry
  HERE:  SJMP HERE            ; End of program
```

Q6 **WAP to copy the value 25H to all locations from 2000H to 2100H in the External RAM.**

```
SOLN:  MOV A, #25H            ; Number stored in A as MOVX works only on "A"
       MOV R0, #00H           ; R0 will contain the count
       MOV DPTR, #2000H       ; DPTR contains the Dest address
BACK:  MOVX @DPTR, A          ; Store at Ext RAM location pointed by DPTR
       INC DPTR               ; Inc Dest address
       INC R0                 ; Inc Count
       CJNE R0, #00H, BACK    ; Check if RO has again become 0. If not: repeat
       MOVX @DPTR, A          ; Store at 2100H
HERE:  SJMP HERE              ; End of program
```

In the above program, you must remember that an 8-bit register like R0 will give 2^8 i.e. 256 counts, Hence the extra step after the loop is required.

Special Function Registers

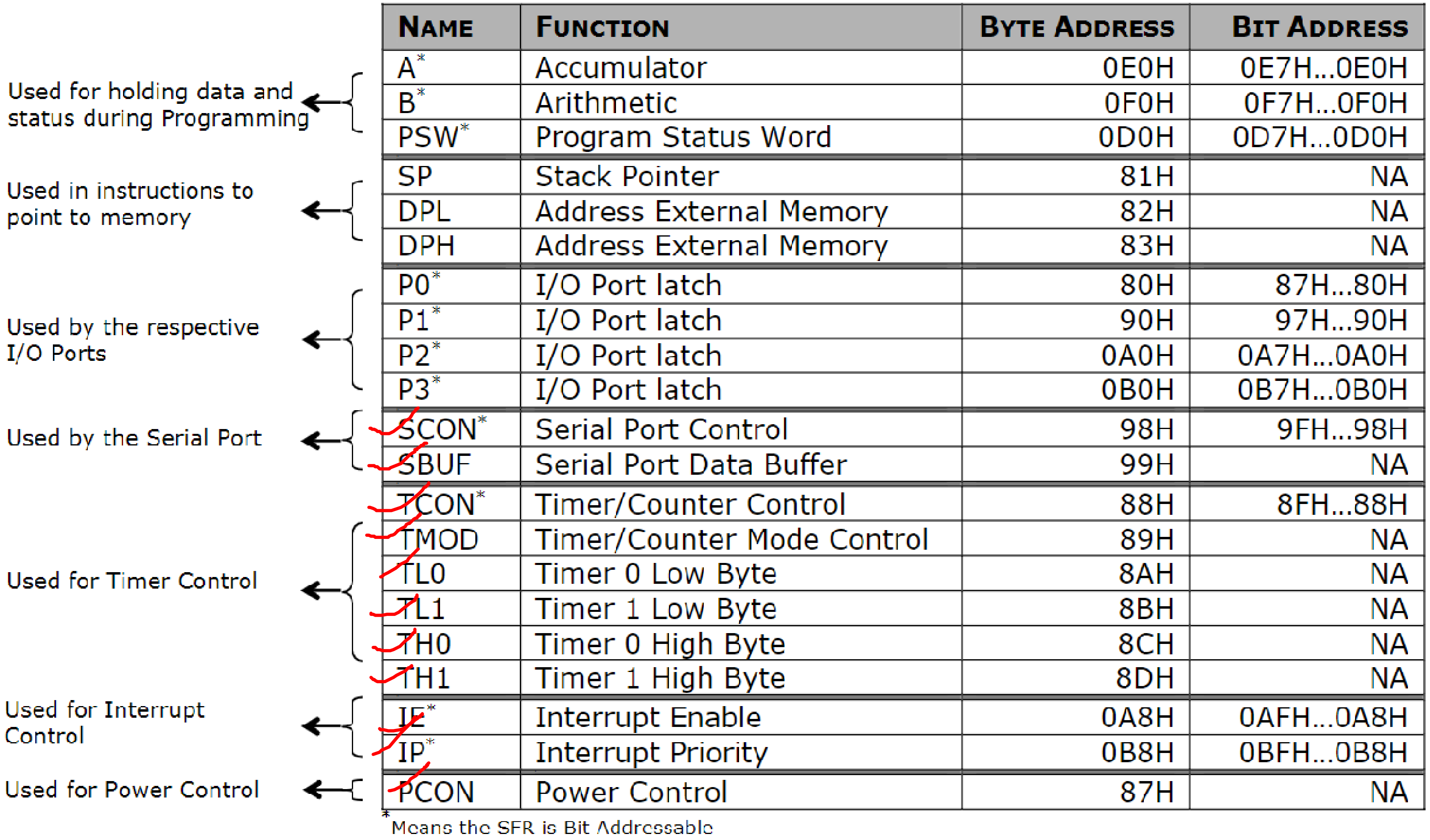
S.No	Microprocessor	Microcontroller
1	Microprocessor acts as a heart of computer system.	Microcontroller acts as a heart of embedded system.
2	It is a processor in which memory and I/O output component is connected externally.	It is a controlling device in which memory and I/O output component is present internally.
3	Since memory and I/O output is to be connected externally. Therefore the circuit is more complex.	Since on chip memory and I/O output component is available. Therefore the circuit is less complex.
4	It cannot be used in compact system. Therefore microprocessor is inefficient.	It can be used in compact system. Therefore microcontroller is more efficient.
5	A microprocessor having a zero status flag.	A microcontroller has no zero flag.
6	It is mainly used in personal computers.	It is mainly used in washing machines, air conditioners etc.
7	It doesn't consist of RAM, ROM, I/O ports. It uses its pins to interface to peripheral devices.	It consists of CPU, RAM, ROM, I/O ports.
8	Its power consumption is high because it has to control the entire system.	It is built with CMOS technology, which requires less power to operate.
10	Microprocessors are used for big applications	Microcontrollers a10re used to execute a single task within an application.

Registers

- General purpose and special purpose registers
- General purpose register : Banks, bit addressable area, scratch pad memory
- Special purpose registers:

Special Function Registers (SFRs of 8051)

- 21 8-bit SFR's are there in 8051



Used for holding data and status during Programming

Used in instructions to point to memory

Used by the respective I/O Ports

Used by the Serial Port

Used for Timer Control

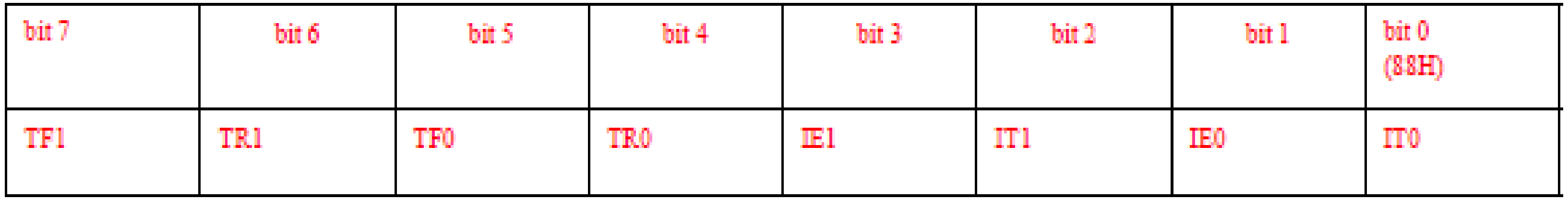
Used for Interrupt Control

Used for Power Control

NAME	FUNCTION	BYTE ADDRESS	BIT ADDRESS
A*	Accumulator	0E0H	0E7H...0E0H
B*	Arithmetic	0F0H	0F7H...0F0H
PSW*	Program Status Word	0D0H	0D7H...0D0H
SP	Stack Pointer	81H	NA
DPL	Address External Memory	82H	NA
DPH	Address External Memory	83H	NA
P0*	I/O Port latch	80H	87H...80H
P1*	I/O Port latch	90H	97H...90H
P2*	I/O Port latch	0A0H	0A7H...0A0H
P3*	I/O Port latch	0B0H	0B7H...0B0H
SCON*	Serial Port Control	98H	9FH...98H
SBUF	Serial Port Data Buffer	99H	NA
TCON*	Timer/Counter Control	88H	8FH...88H
TMOD	Timer/Counter Mode Control	89H	NA
TL0	Timer 0 Low Byte	8AH	NA
TL1	Timer 1 Low Byte	8BH	NA
TH0	Timer 0 High Byte	8CH	NA
TH1	Timer 1 High Byte	8DH	NA
IE*	Interrupt Enable	0A8H	0AFH...0A8H
IP*	Interrupt Priority	0B8H	0BFH...0B8H
PCON	Power Control	87H	NA

* Means the SFR is Bit Addressable

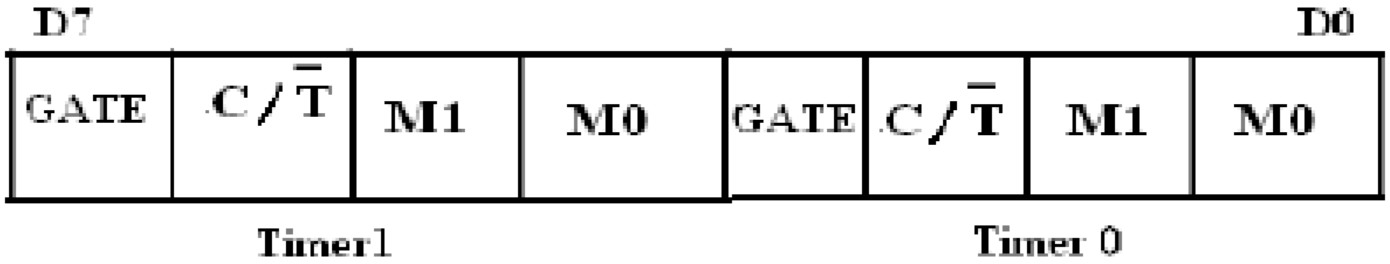
Timer Control Register(TCON)



bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0 (88H)
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- **TF0/TF1:** Timer0/1 overflow flag is set when the timer/ counter overflows, reset by program
- **TR0/TR1:** Timer0/1 run control bit is set to start, reset to stop the timer0/1
- **IE0/IE1:** External Interrupt 0/1 edge detected flag: 1 is set when an external interrupt edge is detected.
- IT0/IT1 External **Interrupt Type (1: falling edge triggered, 0 low level triggered**)

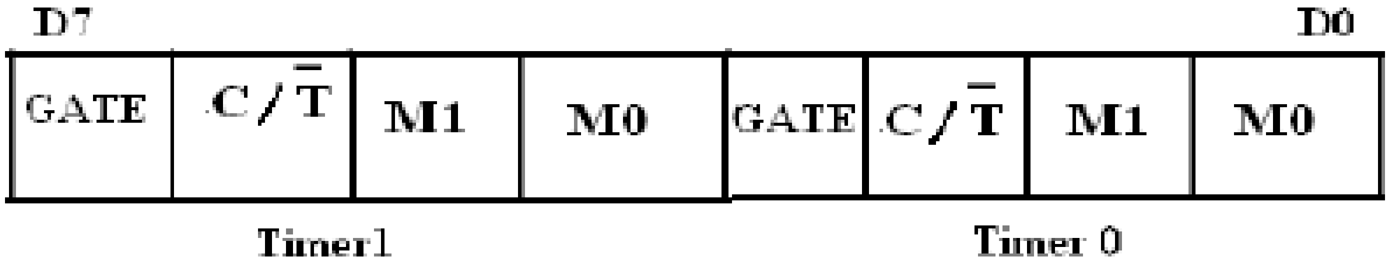
Timer Mode Register(TMOD)



D7							D0
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
Timer1				Timer 0			

- GATE: This bit is used to start or stop the timers by hardware/software .When GATE= 1 ,the timers can be started / stopped by the external sources. When GATE= 0, the timers can be started or stopped by software instructions like SETB TR0 or SETB TR1
- C/T (counter/Timer) : C/T = 0 ,the Timer is used as delay generator and if C/T=1 the timer is used as an event counter.

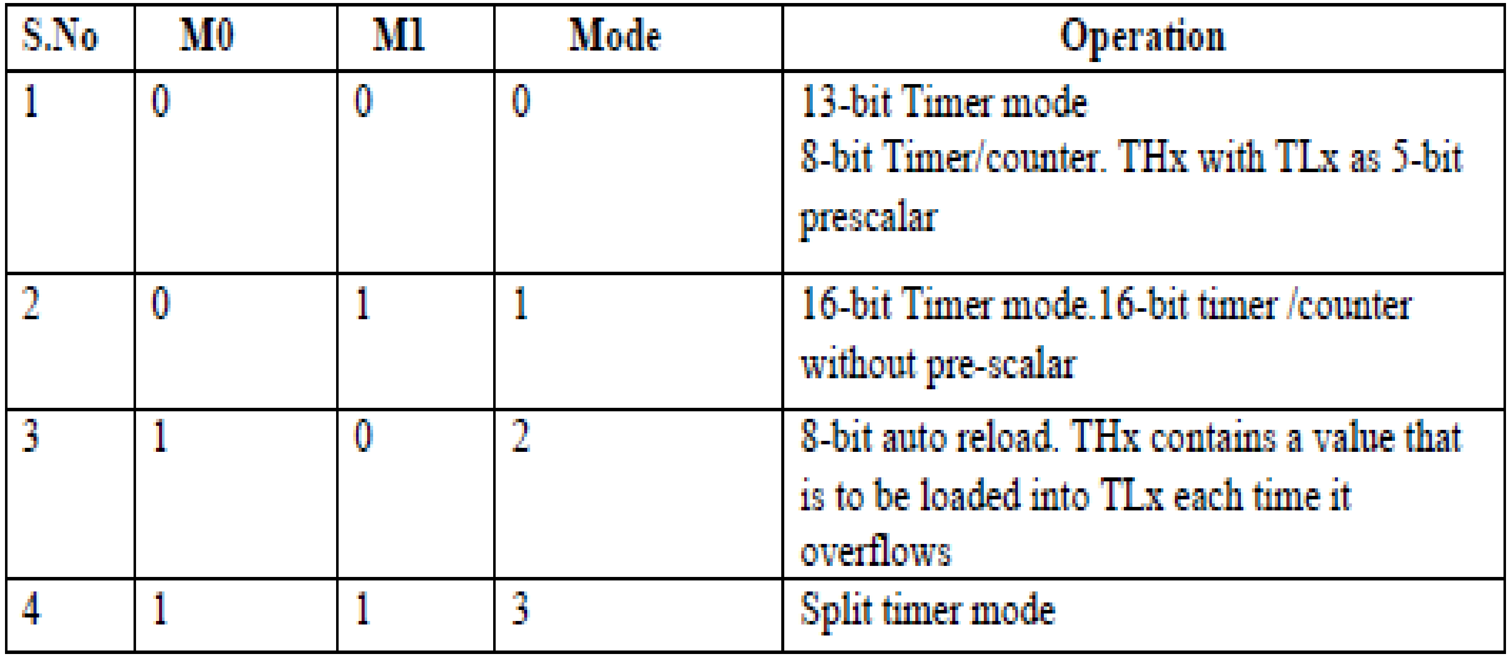
Timer Mode Register(TMOD)



D7 D0

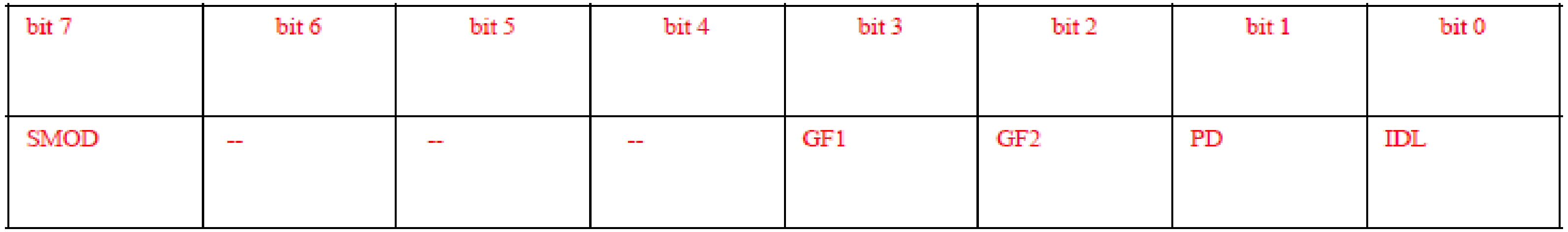
GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0

Timer1 Timer 0



S.No	M0	M1	Mode	Operation
1	0	0	0	13-bit Timer mode 8-bit Timer/counter. THx with TLx as 5-bit prescalar
2	0	1	1	16-bit Timer mode.16-bit timer /counter without pre-scalar
3	1	0	2	8-bit auto reload. THx contains a value that is to be loaded into TLx each time it overflows
4	1	1	3	Split timer mode

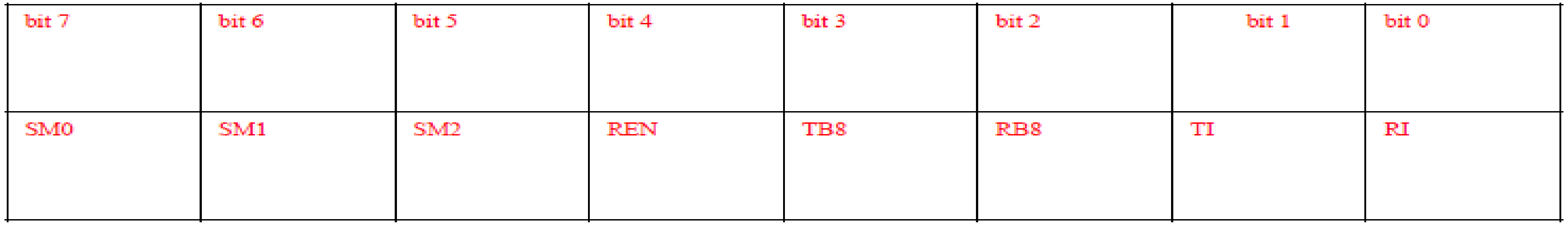
PCON (Power Control Register)



bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
SMOD	--	--	--	GF1	GF2	PD	IDL

- SMOD(serial mode) 1= high baud rate, 0 = low baud rate
- GF1, GF2 flags for free use
- PD: 1= power down mode for CMOS
- IDL: 1= idle mode.
- In **Power Down** mode, the oscillator clock provided to system is OFF i.e. CPU and peripherals clock remains inactive in this mode.
- In **Idle** Mode, only the clock provided to CPU gets deactivated ,whereas peripherals clock will remain active in this mode.
- Hence power saved in power down mode is more than in idle mode.

SCON (Serial Port Control Register)



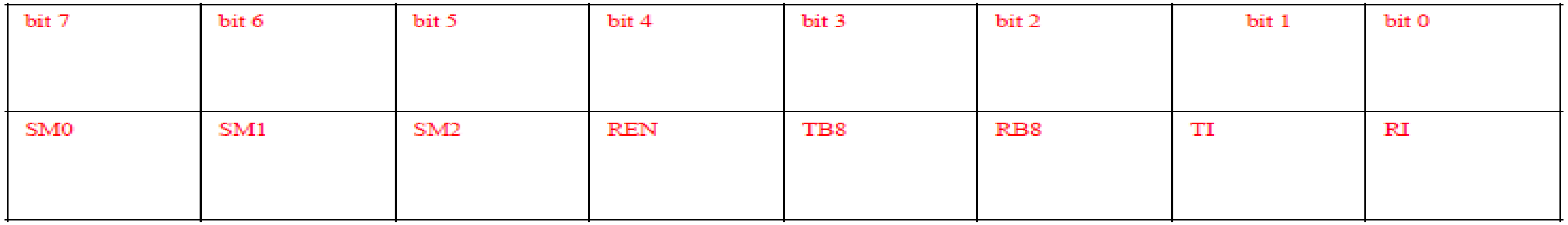
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

- **REN:** Receiver enable is set/reset by program
- **TB8:** This selects 9th bit that will be transmitted in mode2 and mode3
- **RB8:** In mode 2 ,3 this is the 9th data bit that was received.
- **TI:** Transmit Interrupt is set at the end of 8th bit (mode 0) or at the stop bit (other modes) indicating the completion of one byte transmission, reset by program
- **RI:** Receive Interrupt is set at the end of 8th bit (mode 0) or at the stop bit (other modes) indicating the completion of one byte receiving, reset by program
- RI and TI flag in SCON SFR are used to detect the interrupt events.

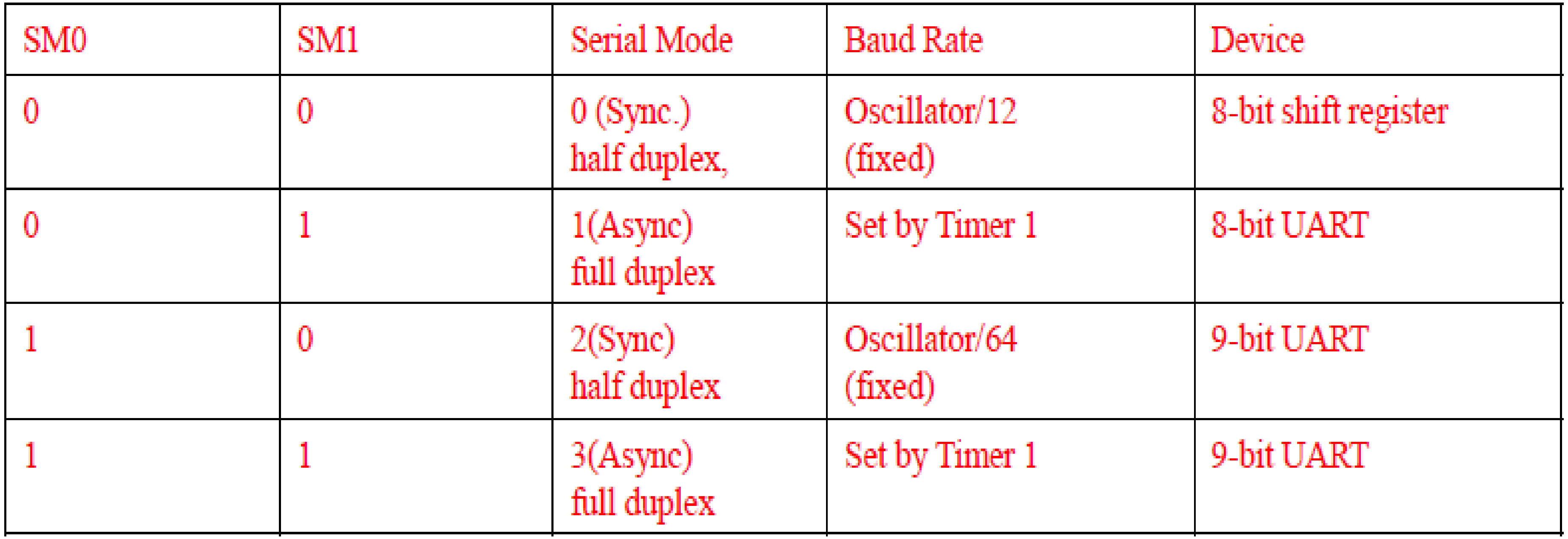
If **RI = 1** then a byte is received at the RxD pin. If **TI = 1** then a byte is

- transmitted from the TxD pin.
- **SM2-**This enables multiprocessor commn feature in mode2 and mode3.

SCON (Serial Port Control Register)

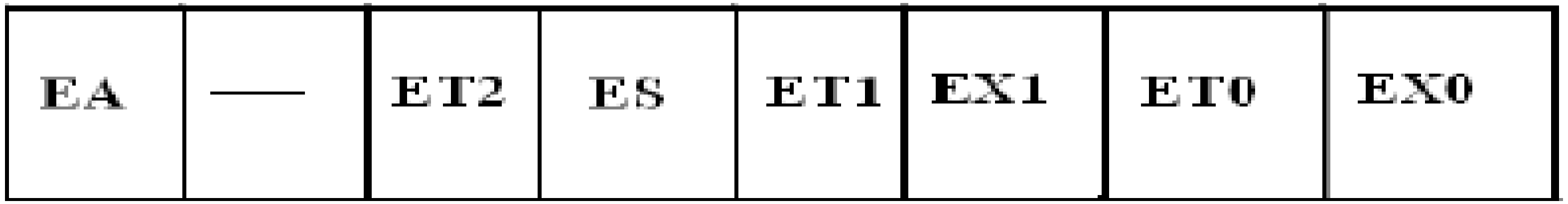


bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI



SM0	SM1	Serial Mode	Baud Rate	Device
0	0	0 (Sync.) half duplex,	Oscillator/12 (fixed)	8-bit shift register
0	1	1(Async) full duplex	Set by Timer 1	8-bit UART
1	0	2(Sync) half duplex	Oscillator/64 (fixed)	9-bit UART
1	1	3(Async) full duplex	Set by Timer 1	9-bit UART

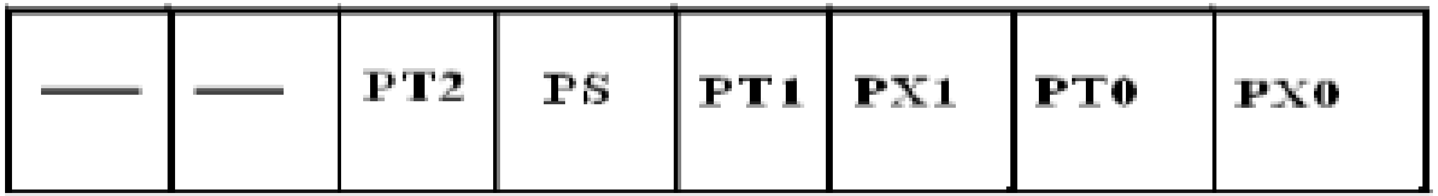
Interrupt Enable Register (IE)



EA	——	ET2	ES	ET1	EX1	ET0	EX0

- EA : Global enable/disable. To enable the interrupts this bit must be set High.
- Undefined-reserved for future use.
- ET2 : Enable /disable Timer 2 overflow interrupt.
- ES : Enable/disable Serial port interrupt.
- ET1 : Enable /disable Timer 1 overflow interrupt.
- EX1 : Enable/disable External interrupt1.
- ET0 : Enable /disable Timer 0 overflow interrupt.
- EX0 : Enable/disable External interrupt 2

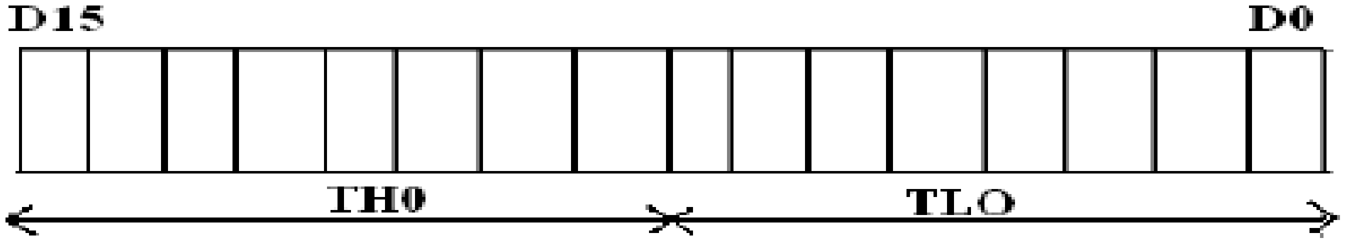
Interrupt Priority Register (IP)

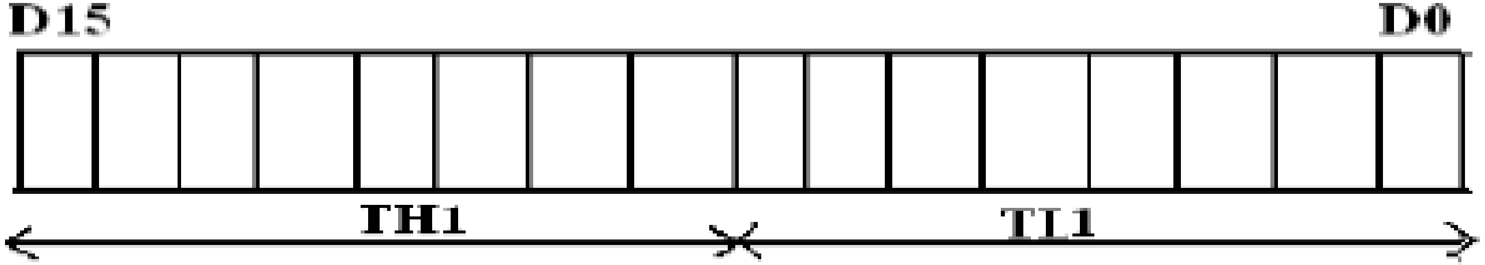


—	—	PT2	PS	PT1	PX1	PT0	PX0

- IP.7: reserved
- IP.6: reserved
- IP.5: Timer 2 interrupt priority bit (8052 only)
- IP.4: Serial port interrupt priority bit
- IP.3: Timer 1 interrupt priority bit
- IP.2: External interrupt 1 priority bit
- IP.1: Timer 0 interrupt priority bit
- IP.0: External interrupt 0 priority bit

TIMER 0 and TIMER 1


D15
D0
TH0
TLO


D15
D0
TH1
TL1