WHAT IS A MICROPROCESSOR?

A **microprocessor** is a controlling unit of a micro-computer wrapped inside a small chip. It performs Arithmetic Logical Unit (ALU) operations and communicates with the other devices connected with it. It is a single Integrated Circuit in which several functions are combined.

WHAT IS MICROCONTROLLER?

A **microcontroller** is a chip optimized to control electronic devices. It is stored in a single integrated circuit which is dedicated to performing a particular task and execute one specific application.

It is specially designed circuits for embedded applications and is widely used in automatically controlled electronic devices. It contains memory, processor, and programmable I/O.

Microprocessor	Microcontroller
Microprocessor is the heart of Computer system.	Micro Controller is the heart of an embedded system.
It is only a processor, so memory and I/O components need to be connected externally	Micro Controller has a processor along with internal memory and I/O components.
Memory and I/O has to be connected externally, so the circuit becomes large.	Memory and I/O are already present, and the internal circuit is small.
You can't use it in compact systems	You can use it in compact systems.
Cost of the entire system is high	Cost of the entire system is low
Due to external components, the total power consumption is high. Therefore, it is not ideal for the devices running on stored power like batteries.	As external components are low, total power consumption is less. So it can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the microcontrollers offer power-saving mode.
It is mainly used in personal computers.	It is used mainly in a washing machine, MP3 players, and embedded systems.
Microprocessor has a smaller number of registers, so more operations are memory-based.	Microcontroller has more register. Hence the programs are easier to write.
Microprocessors are based on Von Neumann model	Micro controllers arc based on Harvard architecture
It is a central processing unit on a single silicon-based integrated chip.	It is a byproduct of the development of microprocessors with a CPU along with other peripherals.

It has no RAM, ROM, Input-Output units, timers, and other peripherals on the chip.	It has a CPU along with RAM, ROM, and other peripherals embedded on a single chip.
It uses an external bus to interface to RAM, ROM, and other peripherals.	It uses an internal controlling bus.
Microprocessor-based systems can run at a very high speed because of the technology involved.	Microcontroller based systems run up to 200MHz or more depending on the architecture.
It's used for general purpose applications that allow you to handle loads of data.	It's used for application-specific systems.
It's complex and expensive, with a large number of instructions to process.	It's simple and inexpensive with less number of instructions to process.

CRITERIA FOR CHOOSING A MICROCONTROLLER

- **Meeting the computing need of the task efficiently and cost effectively**:
- **Power efficiency**.
- **Temperature tolerance** Depending on the environment in which your microcontrollers operate, you may want devices that withstand extreme temperature.
- **Security.**
- **Hardware architecture** A microcontroller's packaging directly influences its size and performance. Dual in-line packaging is the most common type. Small-outline transistors have a small footprint, and quad flat packs take up more areas but less vertical space.
- **Processing power** How much processing power do you require for the task, will a single core processor suffice, or do you need a dual-core? A multicore processor will be significantly faster, but it will also consume more energy. Also, will a graphics processing unit (GPU) be necessary?
- **Memory**. The amount of memory (RAM and ROM) you need will depend on the programs you will be running. More programs need more random access memory (RAM). In addition, a GPU will require not only more RAM but faster read/write time as well.
- **Hardware interface**. The nature of the task will dictate the need for hardware interfaces such as USB, Wi-Fi, Bluetooth, audio, video, or camera.
- **Software architecture**. Some microcontrollers are operable on multiple OSs, and others are not. If you need to scale, it is better to use the same software architecture to increase interoperability
- **Cost**. Microcontrollers fall within a wide price range, from a hundred units for a few dollars to a few dollars per unit. If you want to scale, you need to consider the overall cost versus the individual performance power of a microcontroller.

ADVANTAGES AND DISADVANTAGES OF MICROCONTROLLER

Advantages of the microcontroller:

- The low time required for performing the operation
- It is easy to use, troubleshooting and system maintenance is simple
- At the same time, many tasks can be performed so the human effect can be saved
- The processor chip is very small and flexibility occurs
- Due to their higher integration, cost and size of the system is reduced
- The microcontroller is easy to interface additional RAM, ROM, and I/O port
- Once microcontroller is programmed then they cannot be reprogrammed
- Without any digital parts, it can act as microcomputer
- It is easy to use, troubleshooting and system maintaining is simple

Disadvantages of the microcontroller:

- It is generally used in micro equipment
- It has a more complex structure as compared to microprocessor
- The microcontroller cannot interface a higher power device directly
- It only performed a limited number of executions simultaneously

REGISTERS IN THE 8051 MICROCONTROLLER

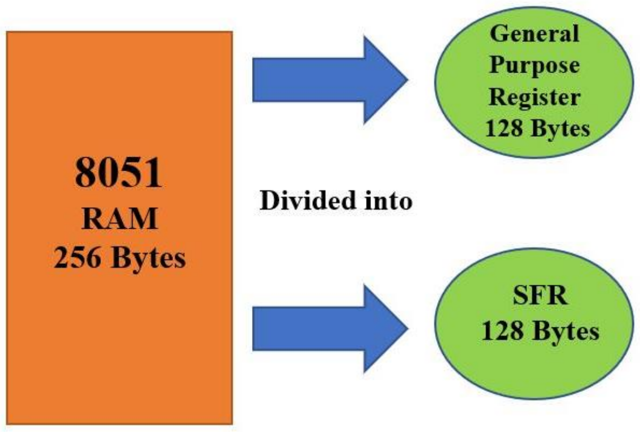
Registers are temporary storage locations inside any processor and a microcontroller that provide a fast way to store data and address of the memory location where the data is to be stored.

If we want to manipulate data within CPU of microcontroller by performing addition, subtraction, and so on, we cannot do that directly in the memory, but to do it you need some registers to process and store the data.

Types of Registers in 8051 Microcontroller

8051 microcontrollers mainly contain two types of registers:

1. General-purpose registers (Byte addressable registers)
2. Special function registers (Bit addressable registers)



8051 Registers

8051 microcontrollers basically consist of 256 bytes of RAM (Random Access Memory), which is divided into two parts, first part contain 128 bytes for general purpose Register these are byte addressable registers and second part contain 128 bytes for special function registers (SFR) memory these are bit addressable registers.

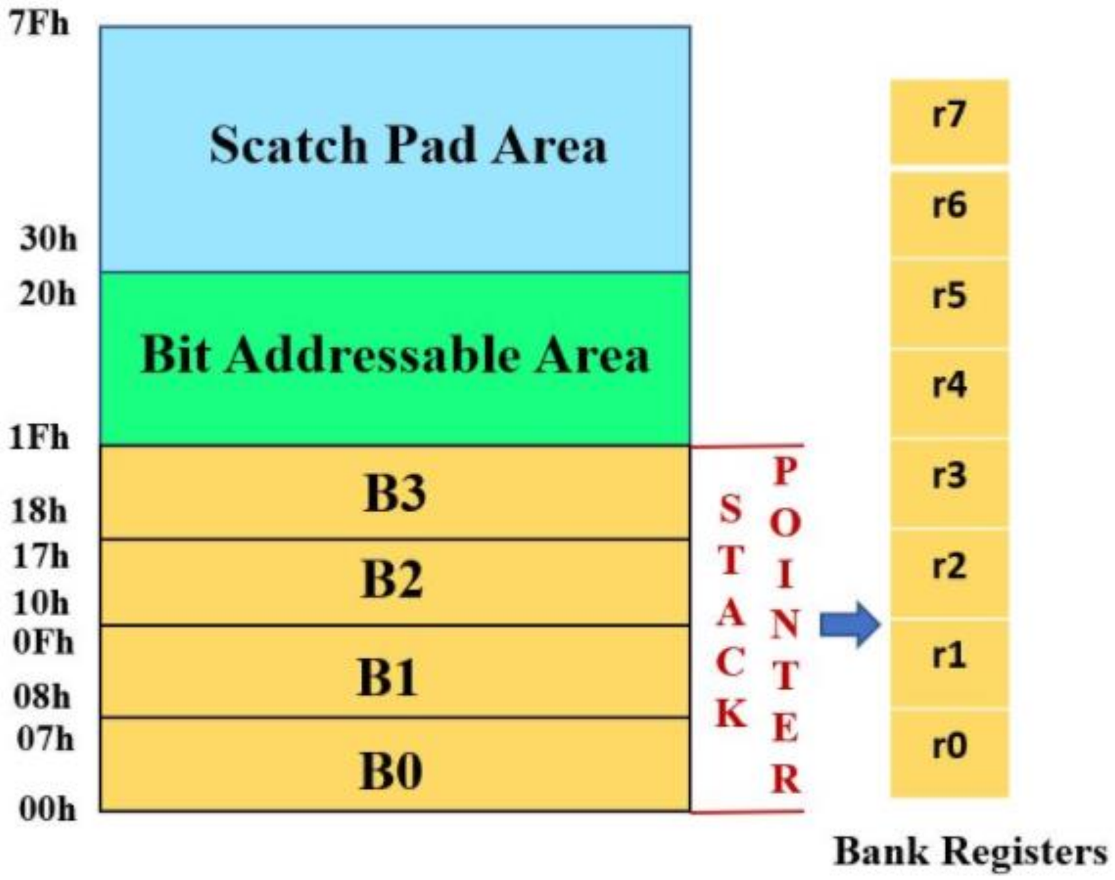
The part of memory which is used for general purpose is called as RAM, and the memory used for Special Function Register (SFR) contains all the peripheral related registers like Accumulator, 'B' register, Timers or Counters, and interrupt related registers.

General Purpose Registers

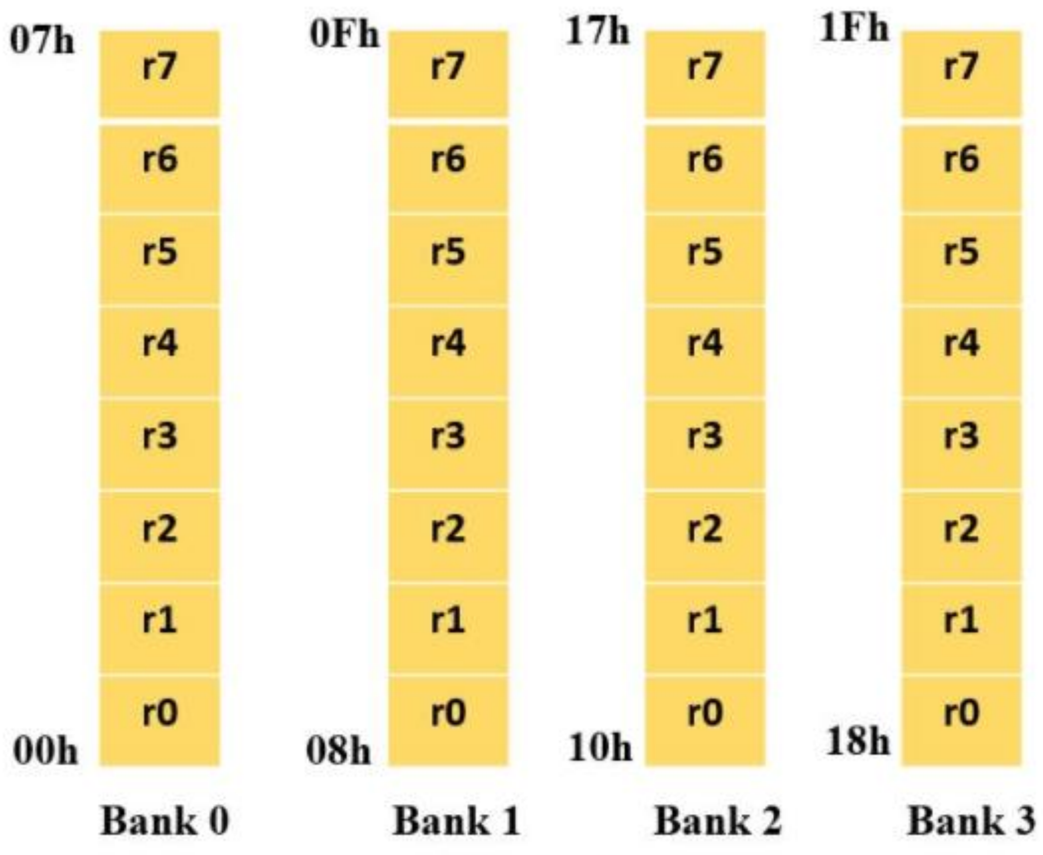
8051 microcontroller has 4 registers bank . These are B0, B1, B2, and B3 stand for Bank0, Bank1, Bank2, Bank3 respectively and each bank contains eight general purpose registers ranging from 'R0' to 'R7'.

These are 32 general purpose registers address from 00h to 1Fh.
The address range of Register Bank 0 (00 h to 07 h)
The address range of Register Bank 1 (08 h to F h)
The address range of Register Bank 2 (10 h to 17 h)
The address range of Register Bank 3 (18 h to 1F h)

These Register Banks are selected with the help of PSW (Program Status Word) Register bits i.e. RS0, RS1.



General Purpose Registers



Banks and Registers

Special Function Registers (SFR's)

In 8051 micro controller there **are 21 Special function registers (SFR)** and this includes Register A, Register B, Processor Status Word (PSW), PCON etc etc. There are 21 unique locations for these 21 special function registers and each of these register is of 1 byte size.

The 21 SFR of 8051 Microcontroller are categorized into seven groups these are:

- **Math or CPU Registers**: A and B Register
- **Status Register**: PSW (Program Status Word) Register
- **Pointer Registers**: DPTR (Data Pointer – DPL, DPH) and SP (Stack Pointer) Registers
- **I/O Port Latches**: P0 (Port 0), P1 (Port 1), P2 (Port 2) and P3 (Port 3) Registers
- **Peripheral Control Registers**: PCON, SCON, TCON, TMOD, IE and IP Registers
- **Peripheral Data Registers**: TL0, TH0, TL1, TH1 and SBUF Registers

Accumulator (A register):

- It is an 8-bit register.
- It holds a data and receives the result of the arithmetic instructions.

- ACC is usually accessed by direct addressing and its physical address is E0H. Accumulator is both byte and bit addressable. if you want to access the second bit (i.e bit 1), you may use E1H and for third bit E2H and so on.

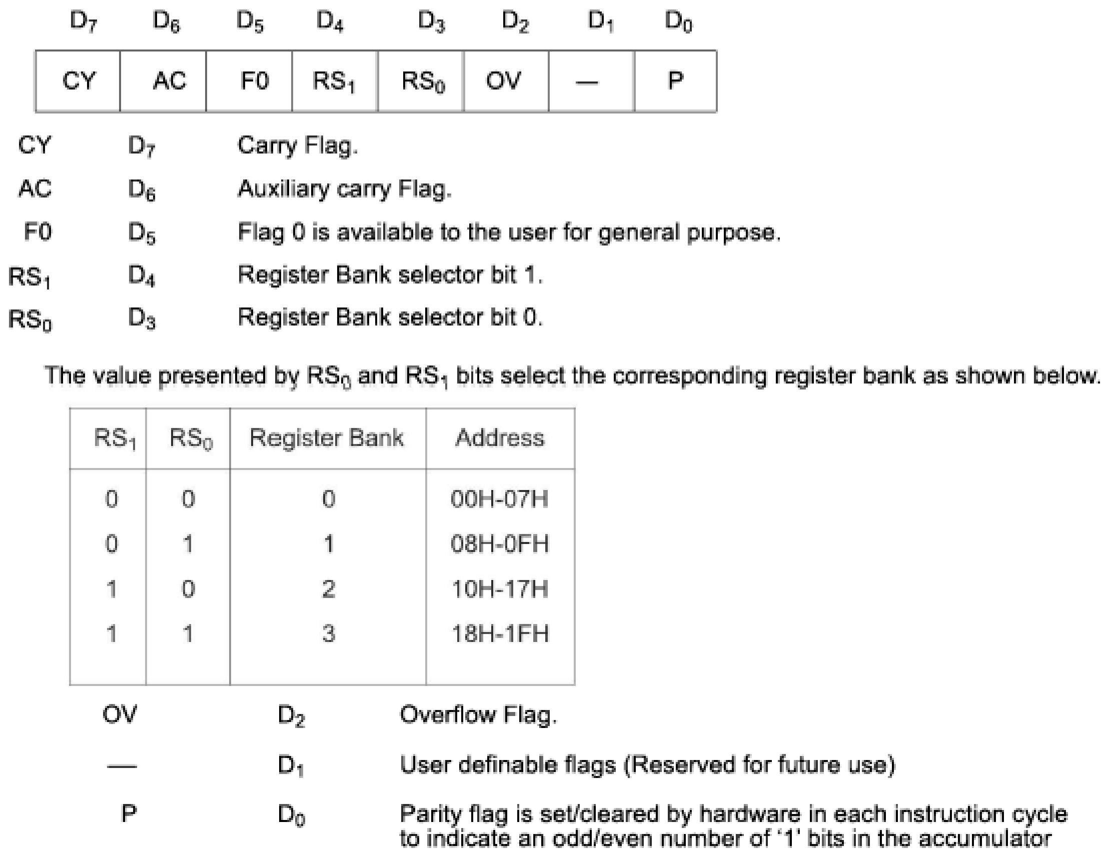
B Register:

- It is 8-bit register.
- It is bit and byte-addressable register.
- You can access 1-bit or all 8-bits by a physical address F0h. Suppose to access a bit 1, we have to use f1.
- The B register is only used for multiplication and division arithmetic operations.

PSW (Program Status Word) Register:

The PSW (Program Status Word) Register is also called as Flag Register. It is one of the important SFRs in 8051 microcontrollers. It is also an 8-bit register. It consists of Flag Bits or status bits that reflect the current state of the CPU. This will help the programmer in checking the condition of the result and also make decisions.

PSW flag register is both bit and byte addressable. The physical address of PSW starts from D0H. The individual bits are accessed by using bit address D1, D2 … D7. The two unused bits are user-defined flags. Four of the flags are called conditional flags, which means that they indicate a condition which results after an instruction is executed. These four are CY (Carry), AC (auxiliary carry), P (parity), and OV (overflow).



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CY	AC	F0	RS_1	RS_0	OV	—	P

CY	D_7	Carry Flag.
AC	D_6	Auxiliary carry Flag.
F0	D_5	Flag 0 is available to the user for general purpose.
RS_1	D_4	Register Bank selector bit 1.
RS_0	D_3	Register Bank selector bit 0.

The value presented by RS_0 and RS_1 bits select the corresponding register bank as shown below.

RS_1	RS_0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

OV	D_2	Overflow Flag.
—	D_1	User definable flags (Reserved for future use)
P	D_0	Parity flag is set/cleared by hardware in each instruction cycle to indicate an odd/even number of '1' bits in the accumulator

Timer/Counter Registers:

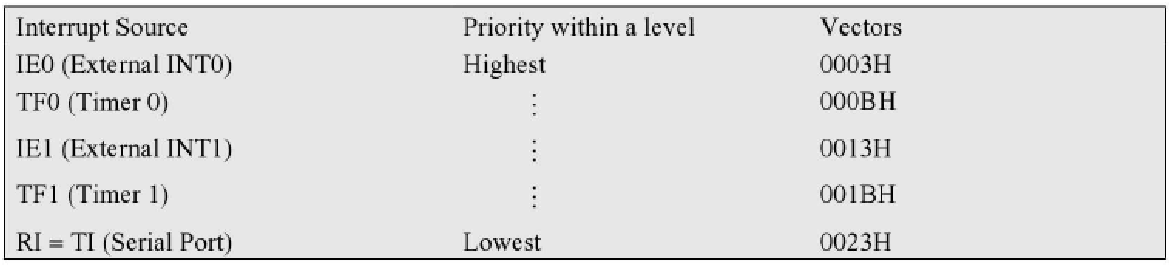
- The 8051 has two 16-bit Programmable timers / counters (Timer 0 – Timer 1).
- Which can be used either as timer to generate a time delay or as counter to count events happening outside the microcontroller.
- The Counters and Timers in 8051 microcontrollers contain two special function registers: TMOD (Timer Mode Register) and TCON (Timer Control Register).

Types of Interrupts in 8051 Microcontroller

The 8051 microcontroller can recognize five different events that cause the main program to interrupt from the normal execution. These five sources of interrupts in 8051are

1. Timer 0 overflow interrupt- TF0
2. Timer 1 overflow interrupt- TF1
3. External hardware interrupt- INT0
4. External hardware interrupt- INT1
5. Serial communication interrupt- RI/TI

The Timer and Serial interrupts are internally generated by the microcontroller, whereas the external interrupts are generated by additional interfacing devices or switches that are externally connected to the microcontroller. These external interrupts can be edge triggered or level triggered. When an interrupt occurs, the microcontroller executes the interrupt service routine so that memory location corresponds to the interrupt that enables it. The Interrupt corresponding to the memory location is given in the interrupt vector table below.



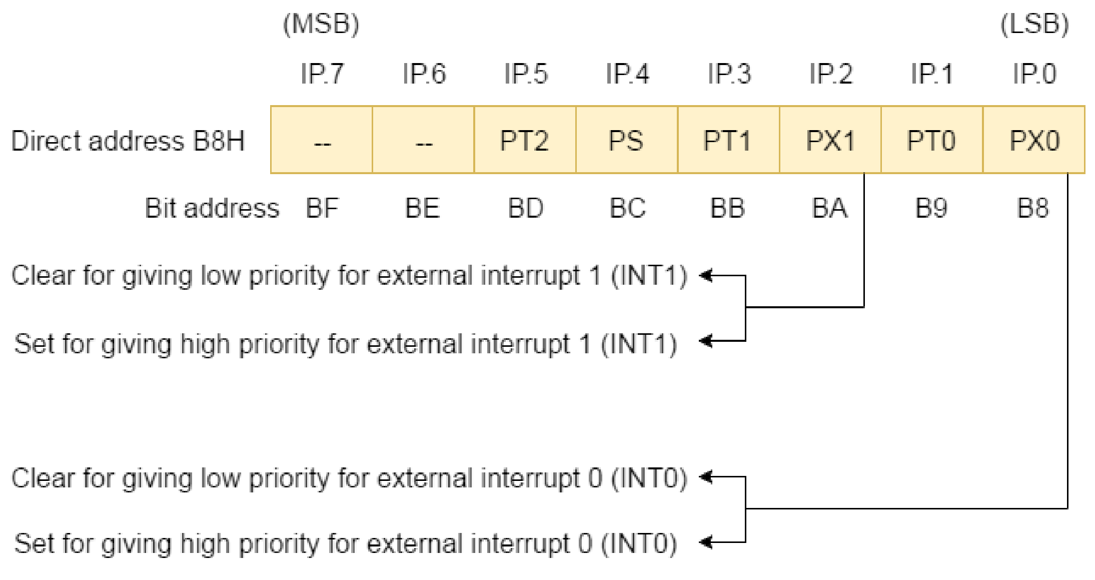
Interrupt Source	Priority within a level	Vectors
IE0 (External INT0)	Highest	0003H
TF0 (Timer 0)	⋮	000BH
IE1 (External INT1)	⋮	0013H
TF1 (Timer 1)	⋮	001BH
RI = TI (Serial Port)	Lowest	0023H

Interrupt Enable (IE) Register: This register is responsible for enabling and disabling the interrupt. It is a bit addressable register in which EA must be set to one for enabling interrupts. The corresponding bit in this register enables particular interrupt like timer, external and serial inputs. In the below IE register, bit corresponding to 1 activates the interrupt and 0 disables the interrupt.

EA	--	--	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts, If EA=0, no interrupt will be acknowledged. If EA=1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
--	IE.6	Not implemented, reserved for future use*.
--	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External interrupt 0.

Interrupt Priority Register (IP): It is also possible to change the priority levels of the interrupts by setting or clearing the corresponding bit in the Interrupt priority (IP) register as shown in the figure. This allows the low priority interrupt to interrupt the high-priority interrupt, but prohibits the interruption by another low-priority interrupt. Similarly, the high-priority interrupt cannot be interrupted. If these interrupt priorities are not programmed, the microcontroller executes in predefined manner and its order is INT0, TF0, INT1, TF1, and SI.

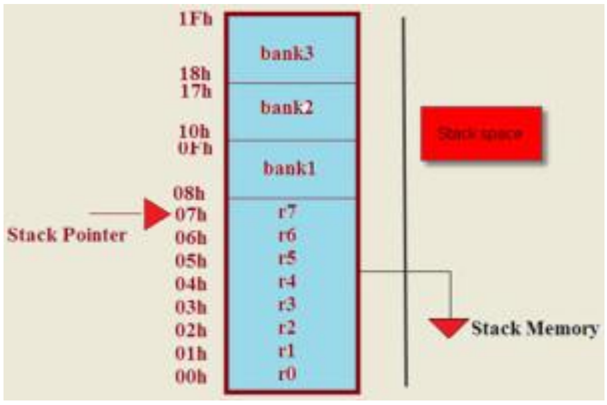

(MSB)
(LSB)
IP.7
IP.6
IP.5
IP.4
IP.3
IP.2
IP.1
IP.0
Direct address B8H
--
--
PT2
PS
PT1
PX1
PT0
PX0
Bit address
BF
BE
BD
BC
BB
BA
B9
B8
Clear for giving low priority for external interrupt 1 (INT1)
Set for giving high priority for external interrupt 1 (INT1)
Clear for giving low priority for external interrupt 0 (INT0)
Set for giving high priority for external interrupt 0 (INT0)

TCON Register: In addition to the above two registers, the TCON register specifies the type of external interrupt to the 8051 microcontroller, as shown in the figure. The two external interrupts, whether edge or level triggered, specify by this register by a set, or cleared by appropriate bits in it. And, it is also a bit addressable register.

Stack Memory Allocation in 8051 Microcontroller

The stack is an area of random access memory (RAM) allocated to hold temporarily all the parameters of the variables. The stack is also responsible for reminding the order in which a function is called so that it can be returned correctly. Whenever the function is called, the parameters and local variables associated with it are added to the stack (PUSH). When the function returns, the parameters and the variables are removed ("POP") from the stack. This is why a program's stack size changes continuously while the program is running.

The register used to access the stack is called stack pointer register. The stack pointer is a small register used to point at the stack. When we push something into the stack memory, the stack pointer increases.

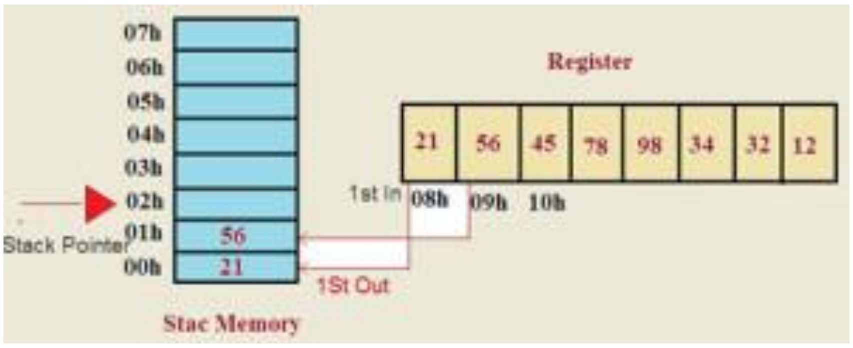


Example

When an 8051 microcontroller power up, the stack pointer contained value is 07, by default, as shown in the above figure. If we perform 'PUSH' operation, then the stack pointer address will be increased and shifted to another register. To avoid this problem, before starting the program, we have to assign a different address location to the stack pointer.

PUSH operation

The 'PUSH' is used for taking the values from any register and storing in the starting address of the stack pointer, i.e., 00h by using 'PUSH' operation. And, for the next 'PUSH', it increments +1, and stores the value in the next address of the stack pointer, i.e., 01h.



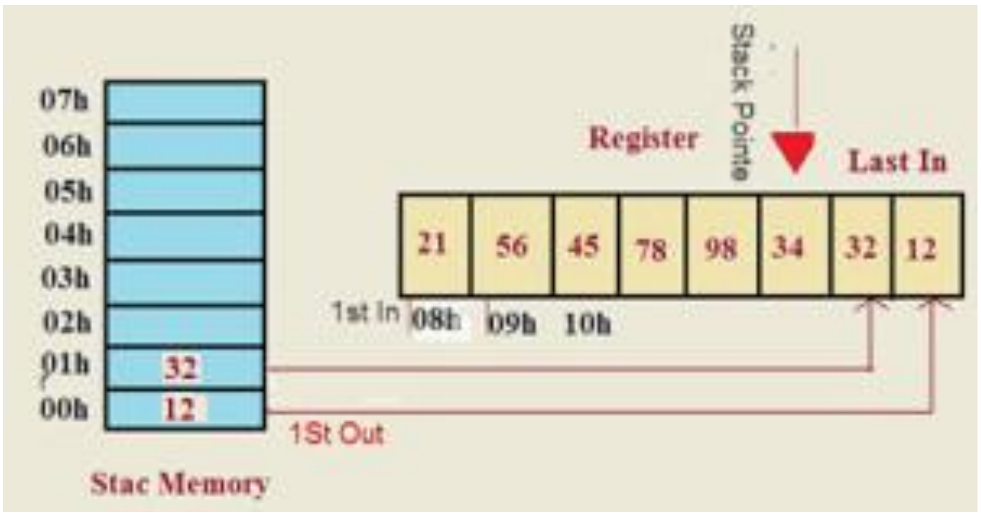
Example: WAP in assembly language for PUSH operation

0000h
MOV 08h, #21h

```
MOV 09h, #56h
PUSH 00h
PUSH 01h
END
```

POP Operation

It is used for placing the values from the stack pointer's maximum address to any other register's address. If we use this 'POP' again, then it decrements by 1, and the value stored in any register is given as 'POP'.



POP operation means 'Last in First out'.

```
000H
MOV 00H, #12H
MOV 01H, #32H
POP 1FH
POP 0EH
END
```

<u>**ADDRESSING MODES OF 8051**</u>

The way of accessing data is called addressing mode. The CPU can access the data in different ways by using addressing modes. The 8051 microcontroller consists of five addressing modes such as:

Immediate Addressing Mode:

In this addressing mode, the source must be a value that can be followed by the '#' and destination must be SFR

registers, general purpose registers and address. It is used for immediately storing the value in the memory registers.

Syntax:

MOV A, #20h //A is an accumulator register, 20 is stored in the A//

MOV R0,#15 // R0 is a general purpose register; 15 is stored in the R0 register//

MOV P0, #07h //P0 is a SFR register;07 is stored in the P0//

MOV 20h,#05h //20h is the address of the register; 05 stored in the 20h//

MOV DPTR,#4532H // DPTR=4532H

Register Addressing Mode:

In this addressing mode, the source and destination must be a register, but not general purpose registers. So the data is not moved within the general purpose bank registers.

Syntax:

MOV A, B; // A is a SFR register, B is a general purpose register//

MOV R0, R1 //Invalid instruction, GPR to GPR not possible//

Direct Addressing Mode

In this addressing mode, the source or destination (or both source and destination) must be an address, but not value.

Direct addressing mode In direct addressing mode, the data is in a RAM memory location whose address is known, and this address is given as a part of the instruction. Contrast this with the immediate addressing mode in which the operand itself is provided with the instruction.

Syntax:

MOV A,20h // 20h is an address; A is a register//

MOV 00h, 07h // both are addressed of the GPS registers//

Indirect Addressing Mode (Register Indirect mode):

In this addressing mode, the source or destination (or destination or source) must be a indirect address, but not a value

In the register indirect addressing mode, a register is used as a pointer to the data. If the data is inside the CPU, only register R0 and R1 are used for this purpose. In other words, R2-R7cannot be used to hold the address of an operand located in RAM when using this addressing mode. When R0 and R1 are used as pointers, that is, when they hold the address of RAM locations, they must be preceded by the “@” sign.

Ex: - MOV A,@R0 // move contents of RAM location whose address is held by R0 into A.

MOV @R1, B // move contents of B RAM location whose address is held by R1

Base Index Addressing Mode:

This addressing mode is used to read the data from the external memory or ROM memory. All addressing modes cannot read the data from the code memory. The code must read through the DPTR register. The DPTR is used to point the data in the code or external memory.

Syntax:

```
MOVC A, @A+DPTR //C indicates code memory//

MOCX A, @A+DPTR // X indicate external memory//
```

Because the data elements are stored in the program space ROM of the 8051,it uses the instruction MOVC instead of MOV. The 16-bit register DPTR and register "A" are used to form the data element stored in on-chip ROM.

<u>8051 INSTRUCTION SET</u>

DATA TRANSFER INSTRUCTIONS

- These instructions implement a bit,byte or 16-bit data transfer operations between the SRC and DST operands
- Both operands can be internal direct data memory operands
- Both cannot be direct and /or indirect register operands R0 to R7
- Immediate operand can be only a source and not a destination
- Program counter is not accessible

```
MOV A,Rn          //Moves the register to the accumulator

MOV A,direct      //Moves the direct byte to the accumulator

MOV A,@Ri         //Moves the indirect RAM to the accumulator

MOV A,#data       //Moves the immediate data to the accumulator

MOV Rn,A          //Moves the accumulator to the register

MOV Rn,direct     //Moves the direct byte to the register

MOV Rn,#data      //Moves the immediate data to the register

MOV direct,A      //Moves the accumulator to the direct byte

MOV direct,Rn     //Moves the register to the direct byte

MOV direct,direct //Moves the direct byte to the direct byte

MOV direct,@Ri    //Moves the indirect RAM to the direct byte
```

MOV direct,#data //Moves the immediate data to the direct byte

MOV @Ri,A //Moves the accumulator to the indirect RAM

MOV @Ri,direct //Moves the direct byte to the indirect RAM

MOV @Ri,#data //Moves the immediate data to the indirect RAM

MOV DPTR,#data //Moves a 16-bit data to the data pointer

MOVC A,@A+DPTR //Moves the code byte relative to the DPTR to the accumulator (address=A+DPTR)

MOVC A,@A+PC //Moves the code byte relative to the PC to the accumulator (address=A+PC)

MOVX A,@Ri //Moves the external RAM (8-bit address) to the accumulator

MOVX A,@DPTR //Moves the external RAM (16-bit address) to the accumulator

MOVX @Ri,A //Moves the accumulator to the external RAM (8-bit address

MOVX @DPTR,A //Moves the accumulator to the external RAM (16-bit address)

PUSH direct //Pushes the direct byte onto the stack

POP direct //Pops the direct byte from the stack

XCH A,Rn //Exchanges the register with the accumulator

XCH A,direct //Exchanges the direct byte with the accumulator

XCH A,@Ri //Exchanges the indirect RAM with the accumulator

XCHD A,@Ri //Exchanges the low-order nibble indirect RAM with the accumulator

RETI //Returns from interrupt subroutine

ARITHEMETIC INSTRUCTIONS

- These instructions implement arithmetic and logical operations along with increment, decrement and decimal adjust operations
- Accumulator is a compulsory destination operand for two operand instructions
- Immediate operand can be only source and not a destination operand
- Program counter or its part cannot be an operand
- Immediate data cannot be an operand foe INC/DEC instruction

- There are no SUB or Compare instructions .programmer has to implement the substraction without borrow and comparison instructions using SUBB

ADD A,Rn	//Adds the register to the accumulator
ADD A,direct	//Adds the direct byte to the accumulator
ADD A,@Ri	//Adds the indirect RAM to the accumulator
ADD A,#data	//Adds the immediate data to the accumulator
ADDC A,Rn	//Adds the register to the accumulator with a carry flag
ADDC A,direct	//Adds the direct byte to the accumulator with a carry flag
ADDC A,@Ri	//Adds the indirect RAM to the accumulator with a carry flag
ADDC A,#data	//Adds the immediate data to the accumulator with a carry flag
SUBB A,Rn	//Subtracts the register from the accumulator with a borrow
SUBB A,direct	//Subtracts the direct byte from the accumulator with a borrow
SUBB A,@Ri	//Subtracts the indirect RAM from the accumulator with a borrow
SUBB A,#data	//Subtracts the immediate data from the accumulator with a borrow
INC A	//Increments the accumulator by 1
INC Rn	//Increments the register by 1
INC Rx	//Increments the direct byte by 1
INC @Ri	//Increments the indirect RAM by 1
DEC A	//Decrements the accumulator by 1
DEC Rn	//Decrements the register by 1
DEC Rx	//Decrements the direct byte by 1
DEC @Ri	//Decrements the indirect RAM by 1
INC DPTR	//Increments the Data Pointer by 1

MUL AB //Multiplies A and B

DIV AB //Divides A by B

DA A //Decimal adjustment of the accumulator according to BCD code

LOGICAL INSTRUCTIONS

- These instructions implement basic logical operations along with rotate and clear operations
- A is not a compulsory destination operand for logical instructions excluding compliment instruction CPL

ANL A,Rn	//AND register to accumulator
ANL A,direct	//AND direct byte to accumulator
ANL A,@Ri	//AND indirect RAM to accumulator
ANL A,#data	//AND immediate data to accumulator
ANL direct,A	//AND accumulator to direct byte
ANL direct,#data	//AND immediate data to direct register
ORL A,Rn	//OR register to accumulator
ORL A,direct	//OR direct byte to accumulator
ORL A,@Ri	//OR indirect RAM to accumulator
ORL direct,A	//OR accumulator to direct byte
ORL direct,#data	//OR immediate data to direct byte
XRL A,Rn	//Exclusive OR register to accumulator
XRL A,direct	//Exclusive OR direct byte to accumulator
XRL A,@Ri	//Exclusive OR indirect RAM to accumulator
XRL A,#data	//Exclusive OR immediate data to accumulator
XRL direct,A	//Exclusive OR accumulator to direct byte
XORL direct,#data	//Exclusive OR immediate data to direct byte
CLR A	//Clears the accumulator
CPL A	//Complements the accumulator (1=0, 0=1)
SWAP A	//Swaps nibbles within the accumulator

RL A	//Rotates bits in the accumulator left
RLC A	//Rotates bits in the accumulator left through carry
RR A	//Rotates bits in the accumulator right
RRC A	//Rotates bits in the accumulator right through carry
CJNE @Ri,#data,rel	//Compares immediate data to indirect register and jumps if not equal. Short jump.

BOOLEAN OR BIT MANIPULATION INSTRUCTIONS

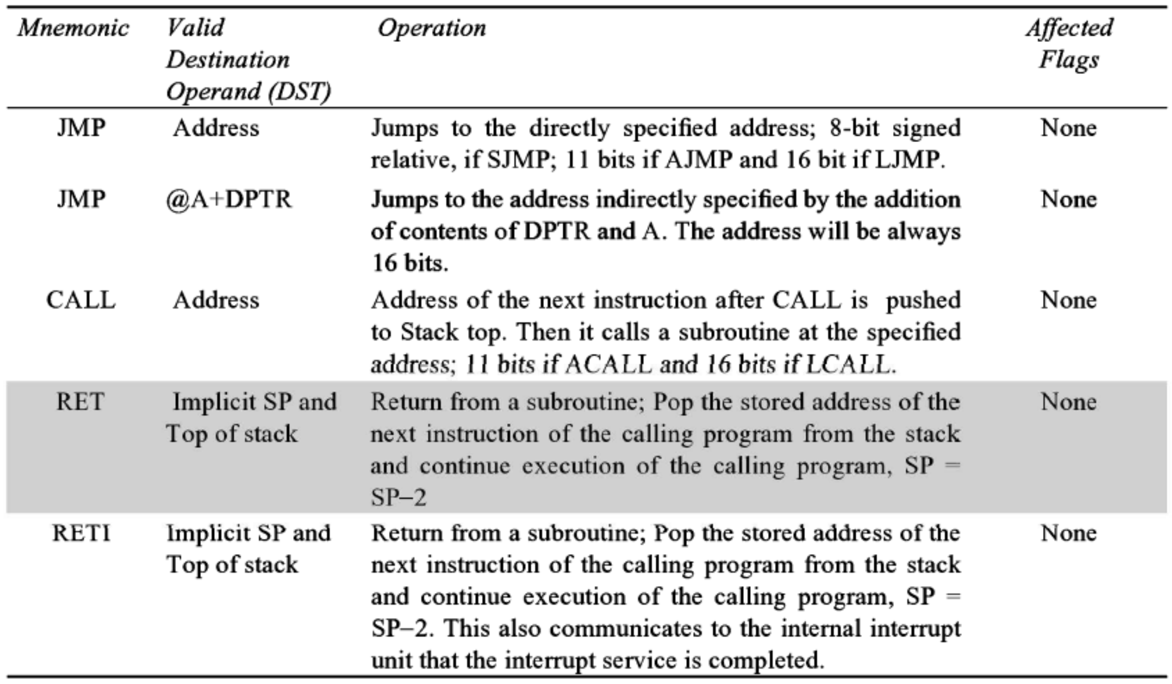
- This group implement Boolean bit operations
- Carry flag(C) as the only allowed destination operand for two operand instructions
- Immediate bit is not allowed as an operand
- The ‘Bit flag’ indicates the respective flag PSW ,if it iis operand of the respective instruction

CLR C	//Clears the carry flag
CLR bit	//Clears the direct bit
SETB C	//Sets the carry flag
SETB bit	//Sets the direct bit
CPL C	//Complements the carry flag
CPL bit	//Complements the direct bit
ANL C,bit	//AND direct bit to the carry flag
ORL C,bit	//OR direct bit to the carry flag
MOV C,bit	//Moves the direct bit to the carry flag
MOV bit,C	//Moves the carry flag to the direct bit

CONTROL TRANFER INSTRUCTIONS

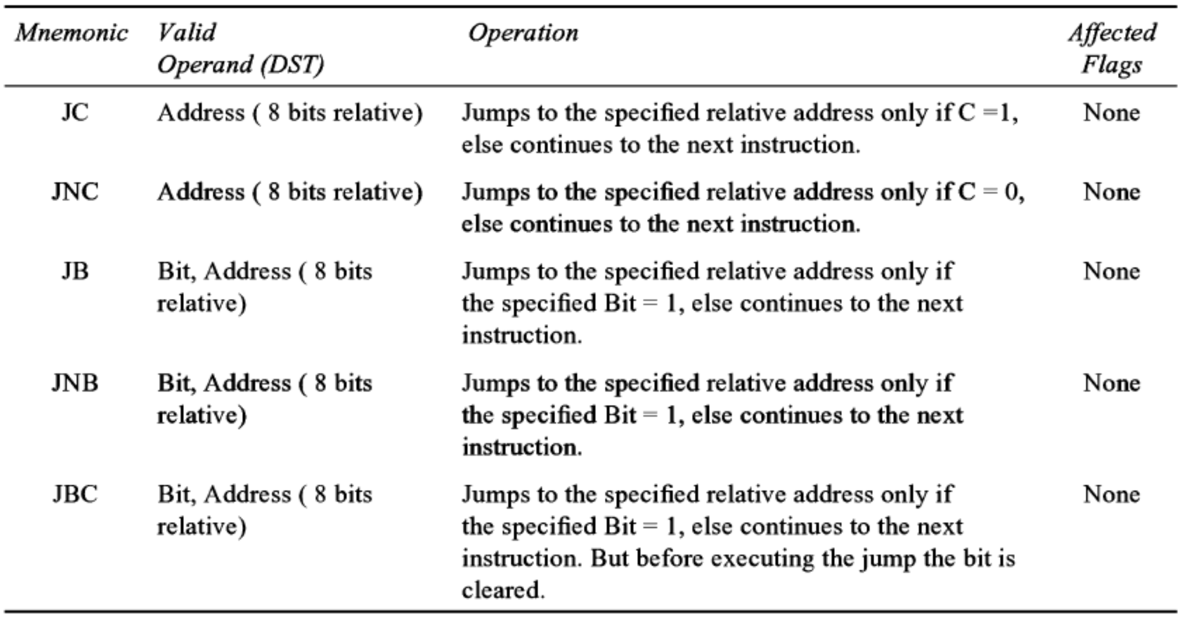
- The control transfer instructions transfer the control of execution or change the sequence of execution conditionally or unconditionally.
- It is divided into two
  - Conditional control transfer
  - Unconditional control transfer

Unconditional Control Transfer Instructions



Mnemonic	*Valid Destination Operand (DST)*	*Operation*	*Affected Flags*
JMP	Address	Jumps to the directly specified address; 8-bit signed relative, if SJMP; 11 bits if AJMP and 16 bit if LJMP.	None
JMP	@A+DPTR	Jumps to the address indirectly specified by the addition of contents of DPTR and A. The address will be always 16 bits.	None
CALL	Address	Address of the next instruction after CALL is pushed to Stack top. Then it calls a subroutine at the specified address; 11 bits if ACALL and 16 bits if LCALL.	None
RET	Implicit SP and Top of stack	Return from a subroutine; Pop the stored address of the next instruction of the calling program from the stack and continue execution of the calling program, SP = SP–2	None
RETI	Implicit SP and Top of stack	Return from a subroutine; Pop the stored address of the next instruction of the calling program from the stack and continue execution of the calling program, SP = SP–2. This also communicates to the internal interrupt unit that the interrupt service is completed.	None

Conditional Jump Instructions



Mnemonic	*Valid Operand (DST)*	*Operation*	*Affected Flags*
JC	Address (8 bits relative)	Jumps to the specified relative address only if C =1, else continues to the next instruction.	None
JNC	Address (8 bits relative)	Jumps to the specified relative address only if C = 0, else continues to the next instruction.	None
JB	Bit, Address (8 bits relative)	Jumps to the specified relative address only if the specified Bit = 1, else continues to the next instruction.	None
JNB	Bit, Address (8 bits relative)	Jumps to the specified relative address only if the specified Bit = 1, else continues to the next instruction.	None
JBC	Bit, Address (8 bits relative)	Jumps to the specified relative address only if the specified Bit = 1, else continues to the next instruction. But before executing the jump the bit is cleared.	None

REVIEW QUESTIONS-1

1. What is a microcontroller? Distinguish between a microcontroller and a microprocessor.
2. Write the criteria for selecting a microcontroller
3. What are the different types of microcontrollers?

REVIEW QUESTIONS-2

4. Explain the following addressing modes of 8051 microcontroller.
   a) Register indirect b) Indexed

5. Describe Internal data memory organization of 8051 microcontroller.
6. Explain PSW of 8051 microcontroller
7. Explain how external memories can be interfaced to 8051 microcontroller
8. Explain the organization of stack in 8051 microcontrollers.
9. Explain the sources of interrupts of 8051 microcontroller.
10. What is the role of IE and IP registers of 8051 microcontroller
11. Describe the following instructions of 8051 microcontroller.
    a) AJMP b) SJMP c) XCHD d) MOVX e) SWAP
12. Write an 8051 assembly language program to count the occurrence of a given byte in a sequence of n bytes.
13. Describe any four control transfer instructions of 8051?
14. Write an 8051 program to count the number of 1s in the binary representation of a given number.
15. What is the use of following 8051 instructions :
    ADDC, SUBB, CPL, RLC and SWAP?
16. Explain about the programmable I/O ports of 8051 microcontroller
17. What is the size of 8051 Stack Pointer (SP)? Discuss the operation of 8051 stack.
18. What is the difference between LCALL and ACALL instructions?
19. Write an 8051 assembly language program to find the largest of ten numbers stored in RAM location 47H onwards. Output the result in port1.
20. Consider four LEDs connected to the lower 4 bits of Port P0 of 8051 microcontroller. Assume that the LEDs shall glow if the corresponding bit is 1. Write an 8051 program which makes the group of LEDs to function as 4-bit Ring Counter. The program should iterate to display the Ring Counter sequence five times continuously and then exit. (Hint: 4bit Ring Counter sequence is 1000, 0100, 0010, 0001)
21. Is "DIV A, R1" a valid instruction? Justify your answer.
22. What are the five different interrupts in 8051?
23. Write an 8051 program to find the sum of digits of an 8bit unsigned decimal number.
24. Write an 8051 based assembly language program to perform addition of two 2x2 matrices.
25. "8051 has bit addressable and byte addressable registers"
    a. What is the difference between the above two types of register
    b. Classify the following 8051 registes into bit addressable or byte-addressable groups :A,P0,P1,SP
26. Write an 8051 program to check whether an 8-bit number stored in internal memory 5FH is a power of 2.If the number is a power of write FFH in register R0?