MICROPROCESSORS AND MICROCONTROLLERS (BECE204L)

DR. NAUSHAD MANZOOR LASKAR

ASSISTANT PROFESSOR SG-1

SCHOOL OF ELECTRONICS ENGINEERING (SENSE)

VELLORE INSTITUTE OF TECHNOLOGY, VELLORE

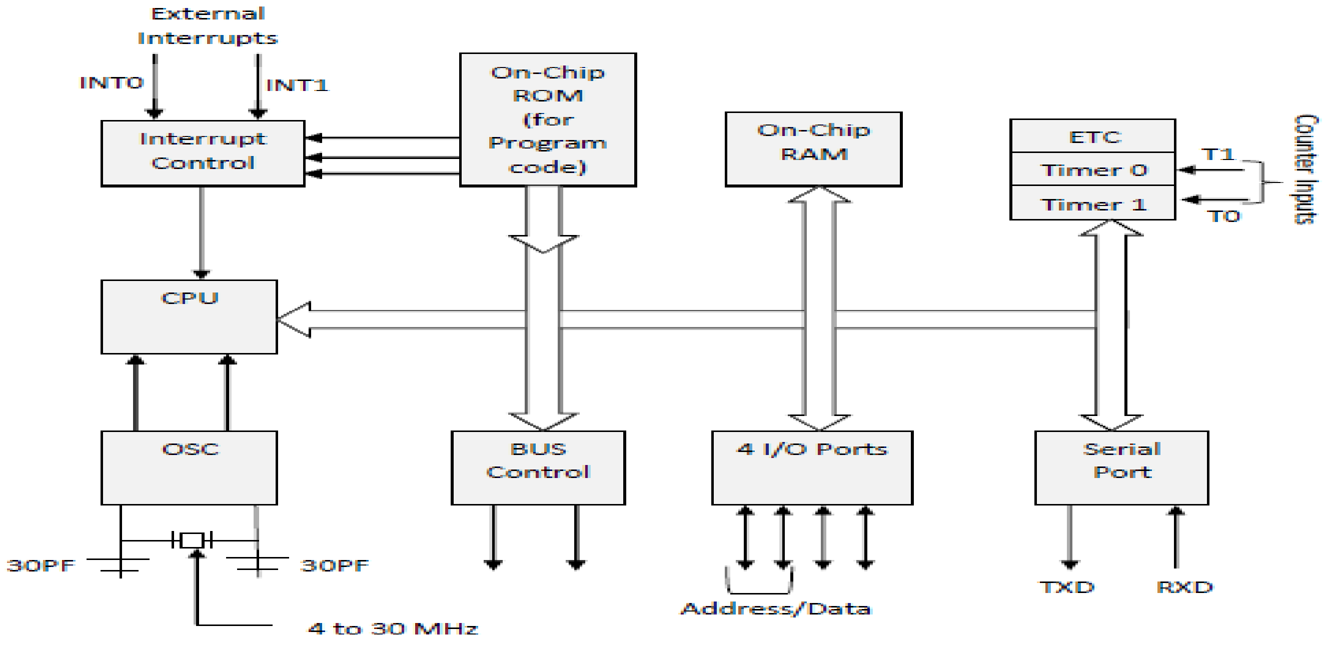
MODULE 3: MICROCONTROLLER ARCHITECTURE INTEL 8051

Introduction to Microcontroller

- A microcomputer made on a single semiconductor chip is called single-chip microcomputer. Since, single chip microcomputers are generally **used in control applications**, they are also called **microcontrollers**
- Contains all essential components of a microcomputer such as CPU, RAM, ROM/EPROM, I/O lines etc.
- Some single chip microcontrollers contain devices to perform specific functions such as DMA channels, **A/D converter,** serial port, **pulse width modulation**, etc

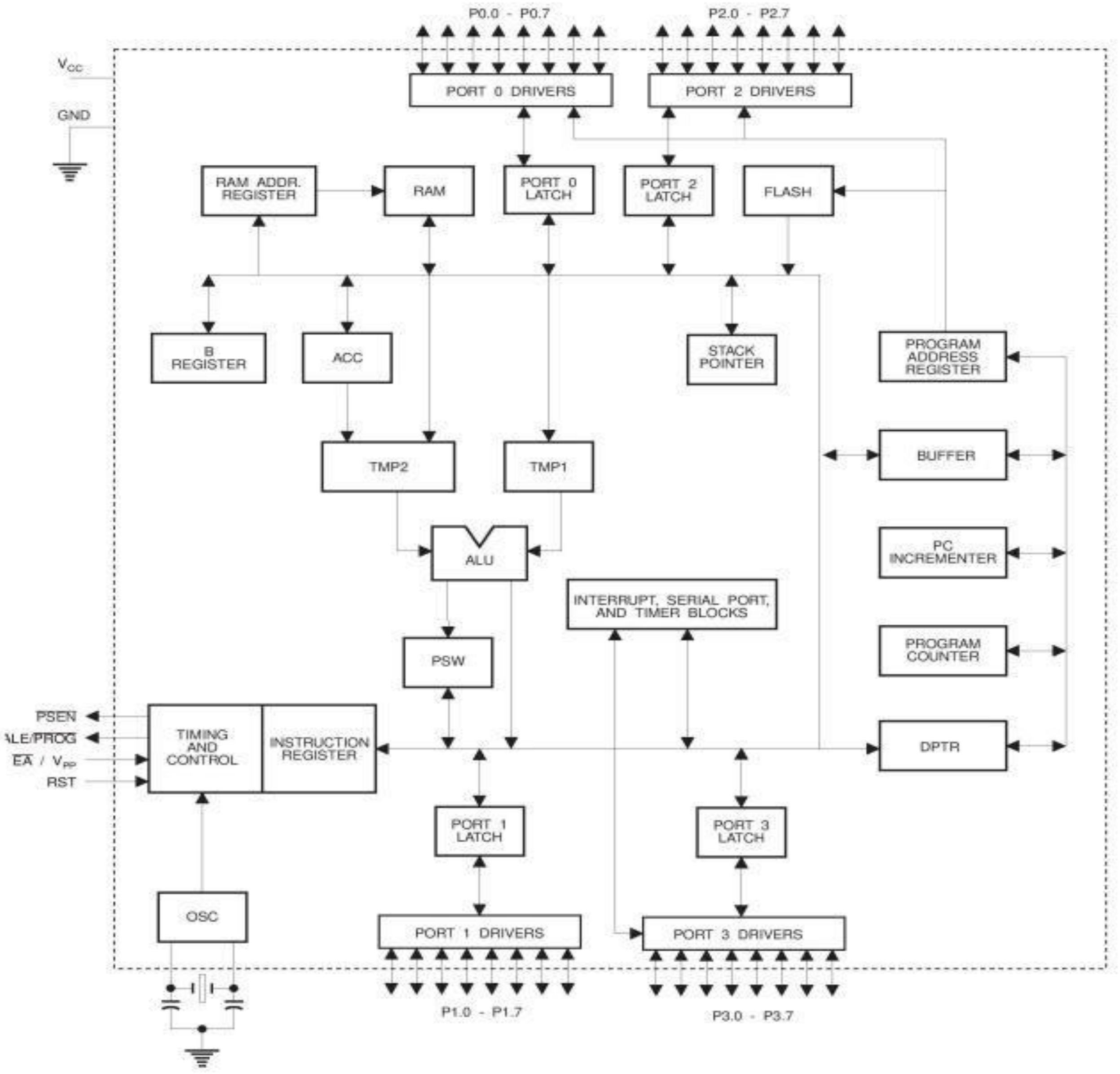
Microprocessor	**Microcontroller**
Just a processor. Memory and I/O components have to be having to be connected externally.	Contains external processor along with internal memory and I/O components.
Since I/O and memory connected externally, the circuit becomes large.	Since I/O and memory present internally, the circuit is small
Power consumption and cost is high	Power consumption and cost is low
Most of the microprocessors do not have power saving modes.	Most of the microcontrollers have power saving mode.
Mainly used in personal computers.	Used mainly in washing machine, MP3 players
Can't be used in compact systems and hence inefficient.	Microcontroller is an efficient technique.

Generalized Architecture of 8051 Microcontroller



System bus connects all the support devices to the CPU. The system bus consists of an 8-bit data bus, a 16-bit address bus and bus control signals. All other devices like program memory, ports, data memory, serial interface, interrupt control, timers, and the CPU are all interfaced together through the system bus

Architecture of AT89C51-Atmel



Crystal Oscillator

The crystal oscillator is the heart of any microcontroller. It generates a master clock of a particular frequency which synchronizes and runs each element on the microcontroller.

Timing and control

The crystal oscillator connects to the timing and control block. This block is responsible for routing the clock signal to the components which need to be operated for the execution of a particular instruction.

CPU

The CPU is the brains of any microcontroller. Apart from handling all the complex mathematical computations of any system, it is also responsible for fetching, decoding, and executing all instructions.

ALU– It is responsible for performing all the logical and mathematical operations. ALU needs data for processing, which it gets from the two temporary registers connected to it

Accumulator– It is a special register that connects to the ALU, and all the results of any computation performed in the CPU are stored there. There is another register know as the B register, which stores the results when the data is too large and cannot be stored in an 8-bit register. This happens during operations like multiplication and division.

Buses

The CPU needs to get data from particular addresses in the RAM. The 8051 has two sets of busses which help in getting data from the RAM. These busses are namely the address bus and data bus. Note: The 8051 has a 16-bit address bus and an 8-bit data bus.

Architecture of AT89C51-Atmel (contd…)

Flash Memory: 8051 uses Harvard architecture. Due to which it has two different memory elements: Flash (ROM) and RAM. Flash memory is where all **your program code** gets stored and tells the microcontroller what to do. The Atmel AT89C51 comes with a 4KB EEPROM, whereas the MCS-51 used an on-chip PROM.

RAM: It has registers that are responsible for storing results that might be needed later during the execution of the program. It includes the various **register banks, special function registers (SFRs),** and free memory space to keep data. The 8051 has a 128 byte RAM space and can be upgraded by using an external RAM.

Program Status Word (PSW)– Manages the various mathematical overheads defined later in the article.

Power Control (PCON)– Selects various power saving modes of 8051.

Serial Control (SCON)– Offers controlling mechanisms for serial communication between various devices and the 8051.

Timer Control (TCON)– Controls the two timers on 8051.

Timer Mode (TMOD)– This **SFR** is used to select the mode of the timer.

Interrupt Priority (IP)– It is used to change the priority of various interrupts.

Interrupt enable (IE)– Used to enable the interrupts on 8051

Stack pointer : A set of registers in the RAM form the stack. This stack of registers is used to store either data or addresses which are required during the execution of a program. To access the data in the stack, a special 8-bit register known as the stack pointer is used. When the microcontroller is powered up, the stack pointer points to the location 07H and grows upwards till the memory location 1FH

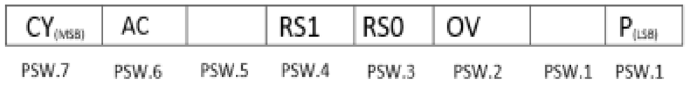
Program status word (PSW): It is a special function register (SFR) that is responsible for managing the various mathematical overheads like carries, auxiliary carries, and much more

DPTR: Data pointer register, as the name suggests, is a register that points to some data. In any computer system, data is kept at a particular address, and the **DPTR is responsible for storing the address.**

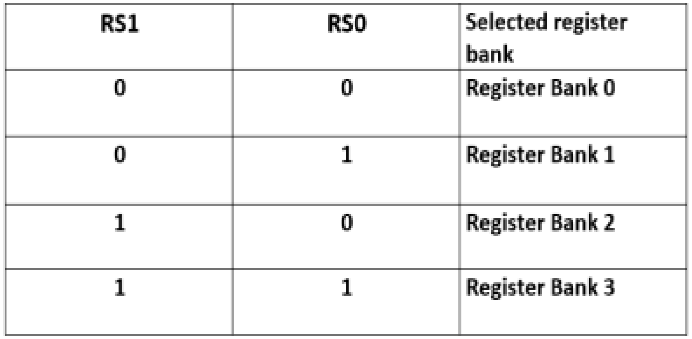
I/O ports: The 8051 has four ports, namely port 0, port 1, port two, and port 3. All these ports are 8-bit ports used for either monitoring a particular device or driving an actuator

Program counter : It is a 16-bit register that is responsible for holding the address of the next instruction, which needs to be executed.

PSW



$CY_{(MSB)}$	AC		RS1	RS0	OV		$P_{(LSB)}$
PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.1



RS1	RS0	Selected register bank
0	0	Register Bank 0
0	1	Register Bank 1
1	0	Register Bank 2
1	1	Register Bank 3

PC Incrementer:

In any microcontroller, the program code is sequentially burned into the flash. So the PC Incrementer is responsible for increasing the address in the program counter register by one. This makes it a point to the next instruction, which needs to be executed.

Timers:

Any electronic device needs to keep time to perform operations in an optimized manner. For this reason, the 8051 has two 16 bit timers, namely T0 and T1.

Pin Diagram of 8051 Microcontroller

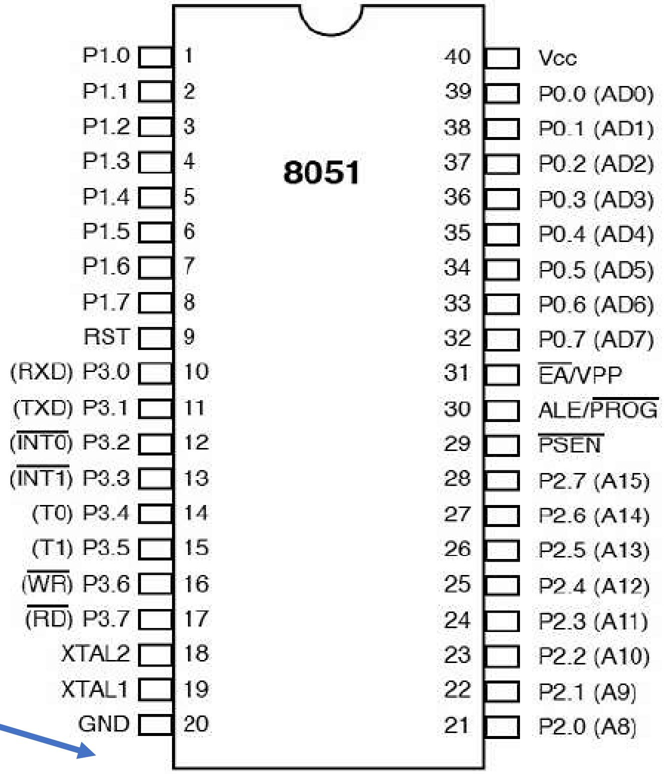
Pin 1-8 (Port1): These are 8-bit bidirectional I/O port with internal pull-up resisters. It does not perform any task; it is just an I/O port

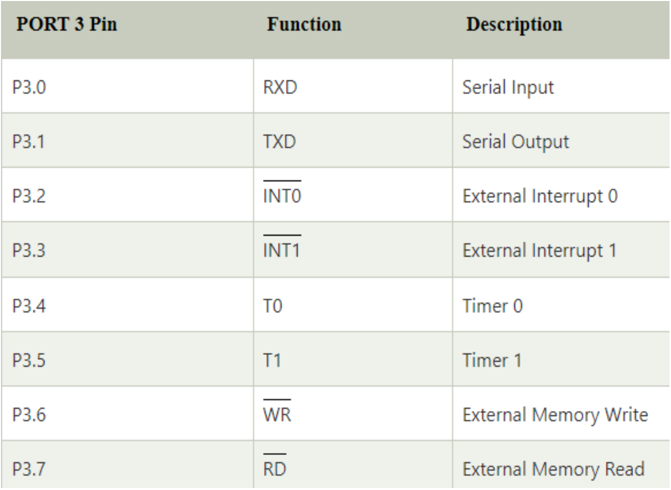
Pin 9 (RST): It is a Reset input pin which is used to reset the microcontroller to its initial position.

Pin 10 to 17 (Port 3): It is also an 8-bit bidirectional I/O port with internal pull-up resisters. Additionally, it performs some special functions:

Pin 18 and 19: It is XTAL1 and XTAL1 pins respectively. These pins are used for connecting an external crystal to get the system clock.

Pin 20 (GND): It is a ground pin. It provides the power supply to the circuit.





PORT 3 Pin	Function	Description
P3.0	RXD	Serial Input
P3.1	TXD	Serial Output
P3.2	$\overline{\text{INT0}}$	External Interrupt 0
P3.3	$\overline{\text{INT1}}$	External Interrupt 1
P3.4	T0	Timer 0
P3.5	T1	Timer 1
P3.6	$\overline{\text{WR}}$	External Memory Write
P3.7	$\overline{\text{RD}}$	External Memory Read

Pin 21 to 28 (Port 2): These pins are bidirectional I/O port. Higher order address bus signals are multiplexed with this bidirectional port.

Pin 29 (PSEN): It is a Program Enable Pin. Using this PSEN pin external program memory can be read.

Pin 30 (ALE/PROG): This pin is the Address Latch Enable pin. Using this pin, external address can be separated from data.

Pin 31 (EA/VPP): Named as external Access Enable Pin (EA). It is used to enable or disable the external memory interfacing.

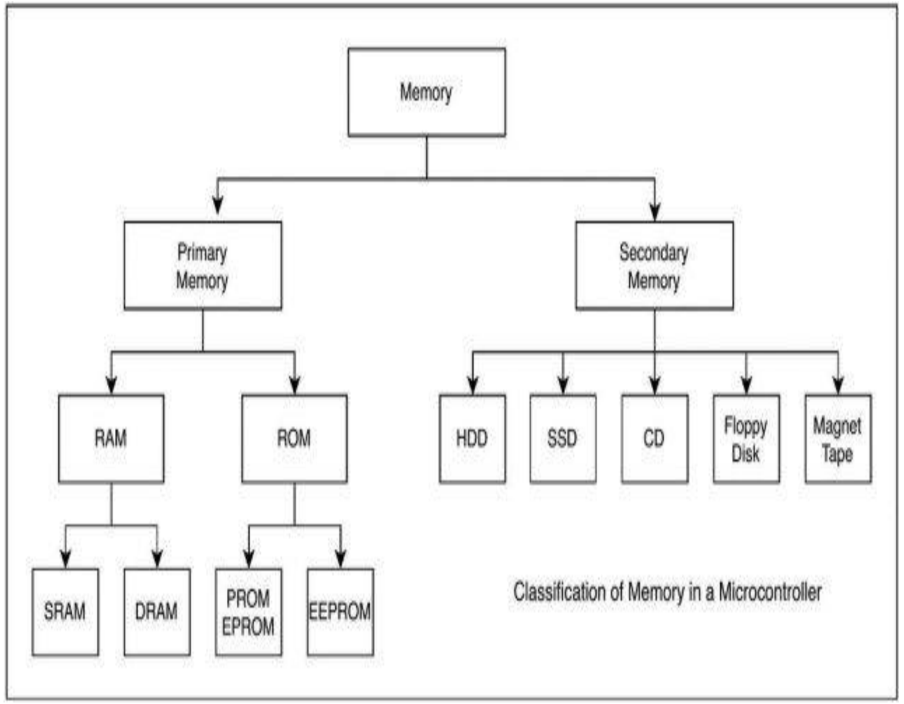
Pin 32 - 39 (Port 0): These are also a bidirectional I/O pins but without any internal pull-ups. Hence, it requires external pins in order to use port 0 pins as I/O port. Lower order data and address bus signals are multiplexed with this port.

Pin 40 (VCC): This pin is used to supply power to the circuit.

Memory Organization

ROM: It is a type of storage that **permanently stores** data on electronic devices. It **contains the code or program that is needed to start a PC**; it performs major input/output tasks and holds programs or software instructions.

RAM: It is the **main memory** in a computer, and it is **much faster to read from and write** to than other kinds of storage, such as a hard disk drive (HDD), solid-state drive (SSD) or optical drive



Classification of Memory in a Microcontroller

Importance of memory in 8051 Microcontroller:

The basic difference between a microprocessor and a microcontroller is the on-chip memory. A microcontroller consists of an **on-chip RAM (Data Memory) and ROM (Program Memory).** The Program Memory is used for permanent saving of the code or program that is to be executed in the PC (Program Counter), whereas the Data Memory is used to temporarily store intermediate results and variables. For a CPU to process the information, it should be stored in either RAM or ROM; these are referred to as primary memory.

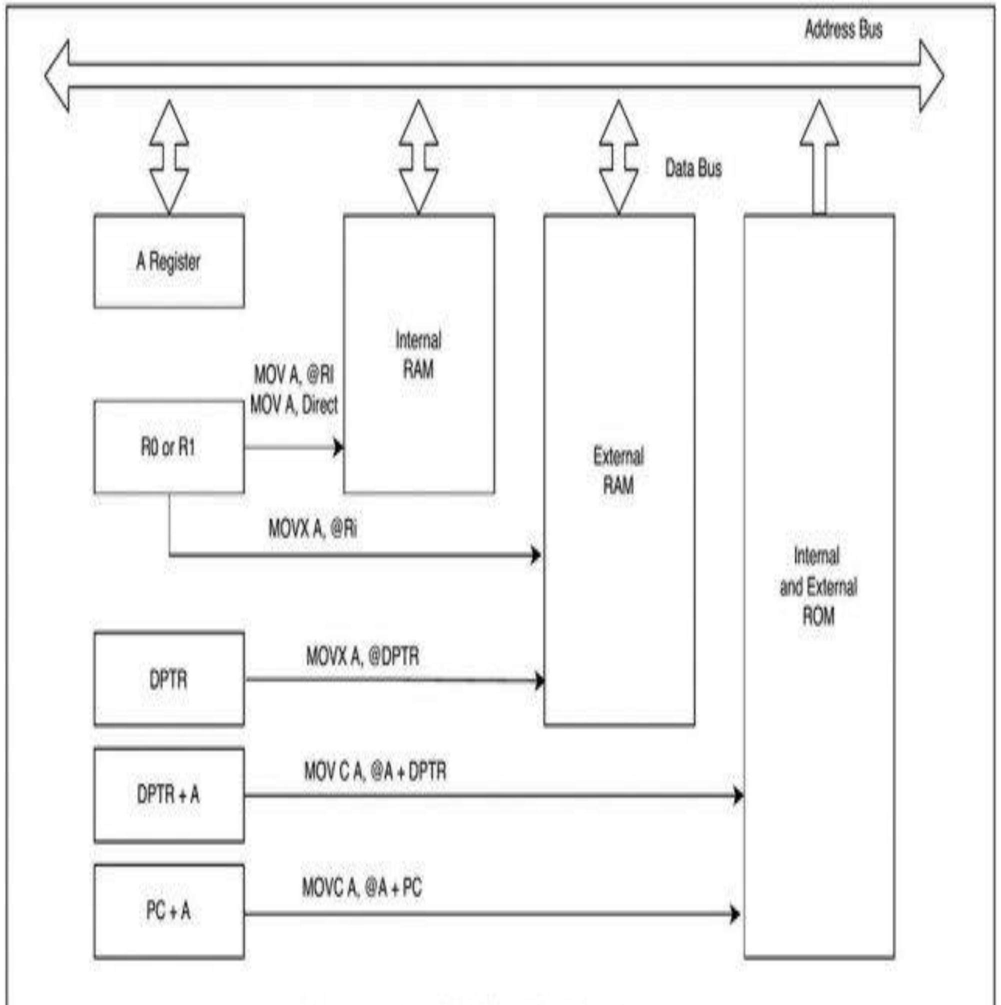


Figure Addressing MOV, MOVC and MOVX instructions

Memory Organization

Program memory organization in 8051: Intel 8051 has an internal/ **built-in ROM of 4KB** and can be extended up to 64KB by using an external program memory. The program memory allocation can be done in two ways depending on the status of the EA pin (External Access Pin), which is an active low pin (i.e., activates when low-signal is provided).

External Program Memory:

1.Internal Program Memory (4KB) i.e. from 0000H to 0FFFH + External Program Memory (60KB) i.e. from 1000H to FFFFH. This mode is selected by making EA = 1. (Fig.1)

2.Total External Program Memory (64KB) i.e., over the entire range of 0000H to FFFFH. This mode is selected by making EA = 0

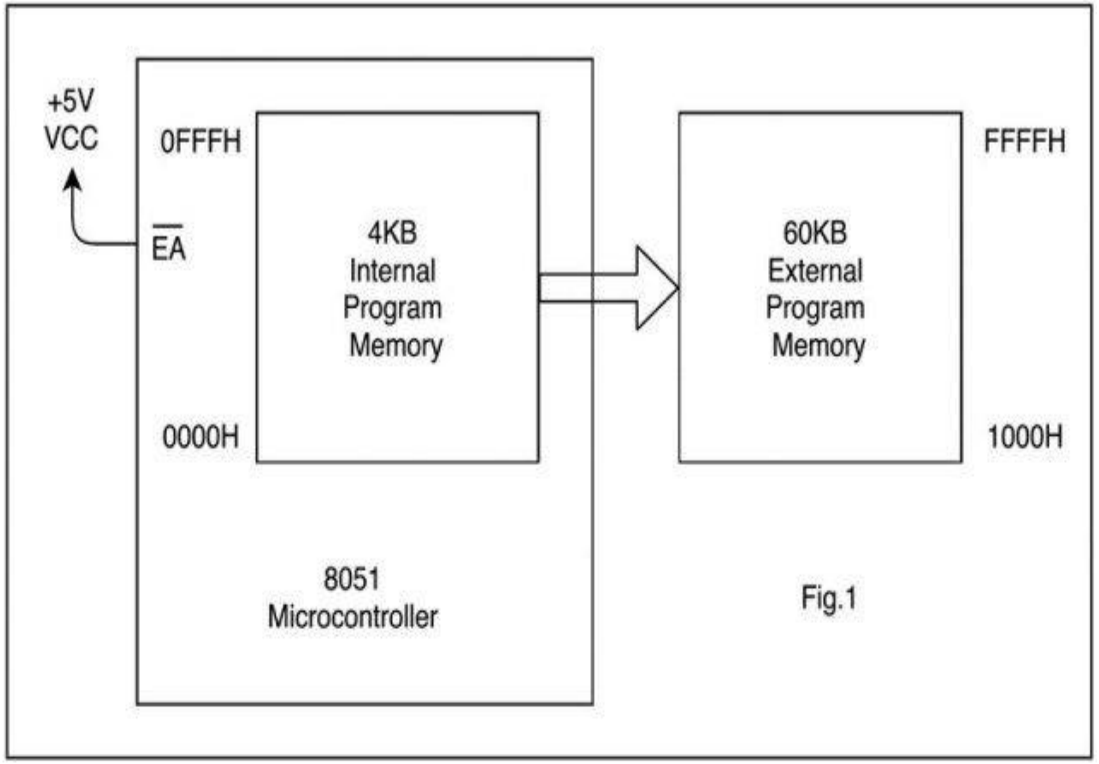


Fig.1

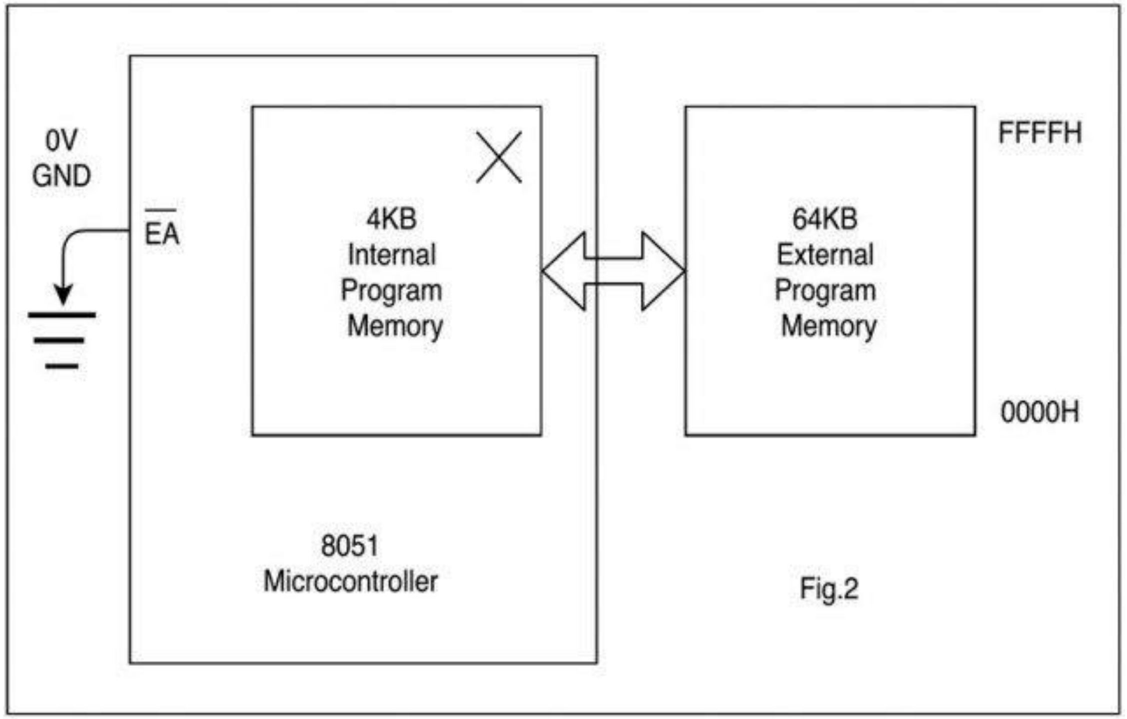


Fig.2

Data memory organization in 8051

- **Data Memory or RAM** is a volatile memory since cutting off power to the IC will result in loss of information or data. RAM is used for temporarily storing the data and the auxiliary results generated during the runtime. Older version of 8051 used to consist built-in 256 bytes of RAM now, and it consists of an additional 128 bytes, which are accessed by indirect addressing

- ***Layout of the RAM in 8051***

The data memory in 8051 is divided into three parts:

1.Lower 128 bytes (00H – 7FH), which are addressed b either Direct or Indirect addressing. Further, the Lower 128 bytes are divided into three parts,

1. Register Banks (Bank 0,1,2,3) from 00H to 1FH – 32 bytes
2. Bit Addressable Area from 20H to 2FH – 16 bytes
3. General Purpose Register (Scratch Pad Area) from 30H to 7FH – 80 bytes

2.Upper 128 bytes (80H – 0FFH) for the Special Function Register (SFRs) which includes I/O ports (P0, P1, P2, P3), Accumulator (A), Timers (THx, TLx, TMOD, TCON, PCON), Interrupts(IE, IP), Serial Communication controls (SBUF, SCON), Program Status Word (PSW). These are addressed using Direct addressing.

3.128 bytes of Additional Memory

- ***Register Banks:*** Registers are something that is used to store information temporarily. Dividing into register banks helps utilize memory effectively. It allows in addressing an entire range of memory using just the register's name

Used to assist in manipulating values and moving data from one memory location to another. Selected using PSW

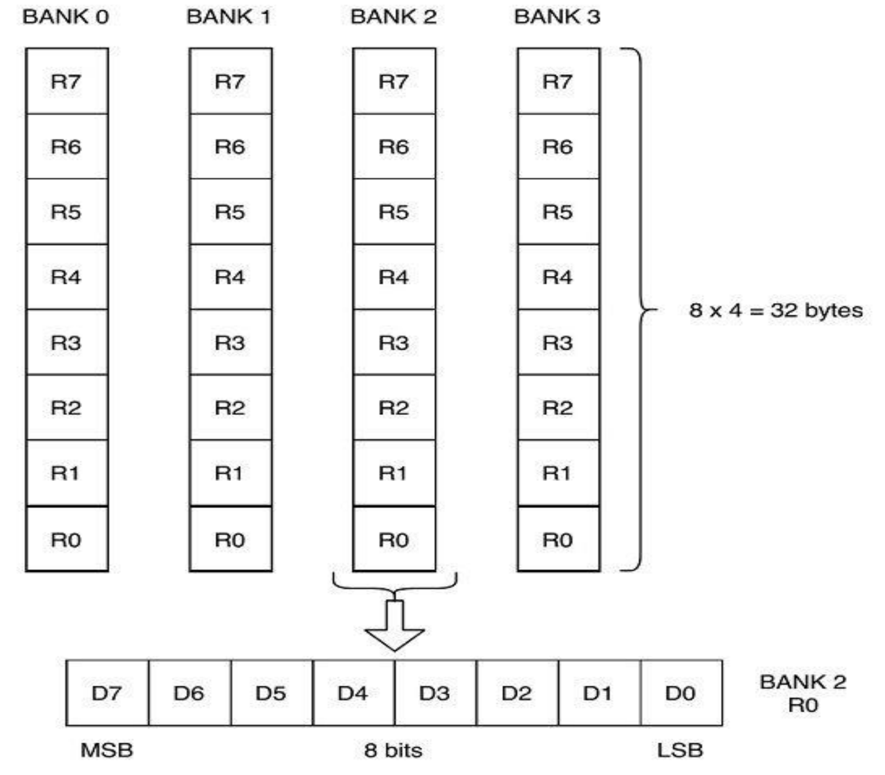


Figure depicting the classification of register banks

Special Function Registers

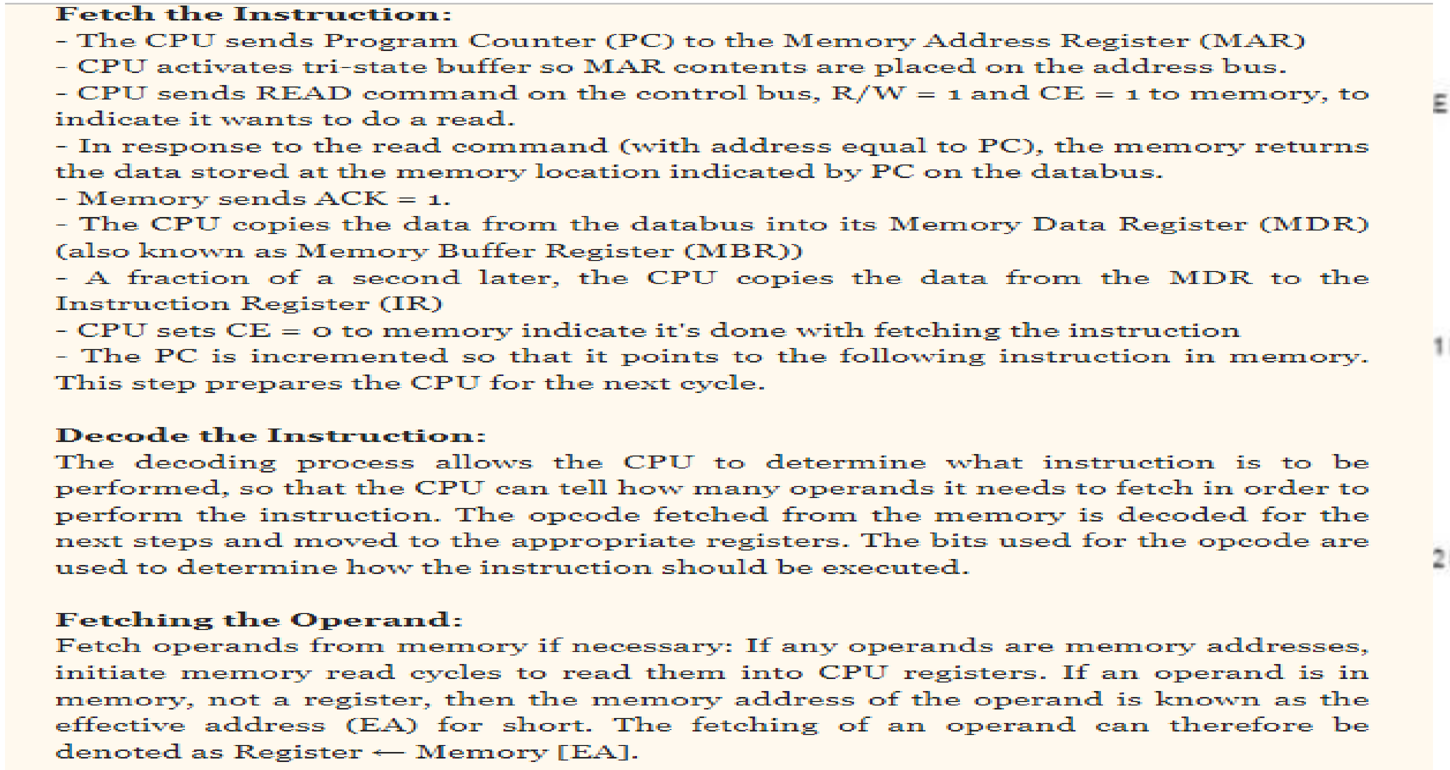
- These registers, abbreviated as SFRs, control various operations of the microcontroller
- The special function register area maintains a set of special function registers that are responsible for managing the various operations occurring in the microcontroller.
- The SFR space (80H to FFH) can be accessed by the programmer using direct addressing to configure them to perform specific tasks, but storing normal program data in this RAM space is not possible.

Some of 8051 SFR are mentioned below:

- **Power control (PCON):** It is responsible for managing the power modes of the 8051 microcontrollers
- **Timer mode (TMOD):** It is responsible for setting the mode of a timer among other things.
- **Timer control (TCON):** It is used to send the control signals for the functioning of the timer
- **Interrupt Enable(IE):** It is used to manage the six interrupts the 8051 microcontroller has. By default, all the interrupts are turned off and need to be turned on by the programmer
- **Interrupt Priority(IP)**
- **Serial control(SCON) :** The 8051 microcontroller can communicate with peripherals using serial communication. To manage this asynchronous communication between devices, the SCON register is used.
- **Input-output related SFRs:** I/O related SFRs help the microcontroller manage ports efficiently for communication with sensors, actuators etc.
- **Port SFRs:**The main job of these 8 bit SFRs is to set the direction of data flow from each of 8 pins of 4 ports
- **Other SFRs:** Accumulator, Stack pointer, PSW, DPTR, SBUF

8051 Machine/Instruction Cycle

- *A Process by which computer receives instructions from memory and determine and carries its course of actions*
- *Sequential Execution*
- *5 steps involved: Instruction Fetch, Inst. Decode, Data Fetch, Inst. Execution, Store Result*



Fetch the Instruction:

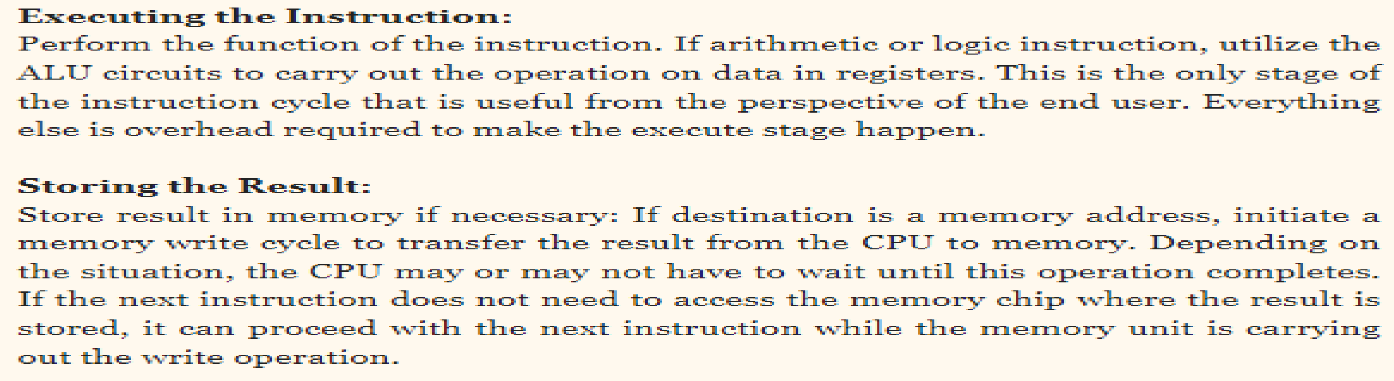
- The CPU sends Program Counter (PC) to the Memory Address Register (MAR)
- CPU activates tri-state buffer so MAR contents are placed on the address bus.
- CPU sends READ command on the control bus, R/W = 1 and CE = 1 to memory, to indicate it wants to do a read.
- In response to the read command (with address equal to PC), the memory returns the data stored at the memory location indicated by PC on the databus.
- Memory sends ACK = 1.
- The CPU copies the data from the databus into its Memory Data Register (MDR) (also known as Memory Buffer Register (MBR))
- A fraction of a second later, the CPU copies the data from the MDR to the Instruction Register (IR)
- CPU sets CE = 0 to memory indicate it's done with fetching the instruction
- The PC is incremented so that it points to the following instruction in memory. This step prepares the CPU for the next cycle.

Decode the Instruction:

The decoding process allows the CPU to determine what instruction is to be performed, so that the CPU can tell how many operands it needs to fetch in order to perform the instruction. The opcode fetched from the memory is decoded for the next steps and moved to the appropriate registers. The bits used for the opcode are used to determine how the instruction should be executed.

Fetching the Operand:

Fetch operands from memory if necessary: If any operands are memory addresses, initiate memory read cycles to read them into CPU registers. If an operand is in memory, not a register, then the memory address of the operand is known as the effective address (EA) for short. The fetching of an operand can therefore be denoted as Register ← Memory [EA].

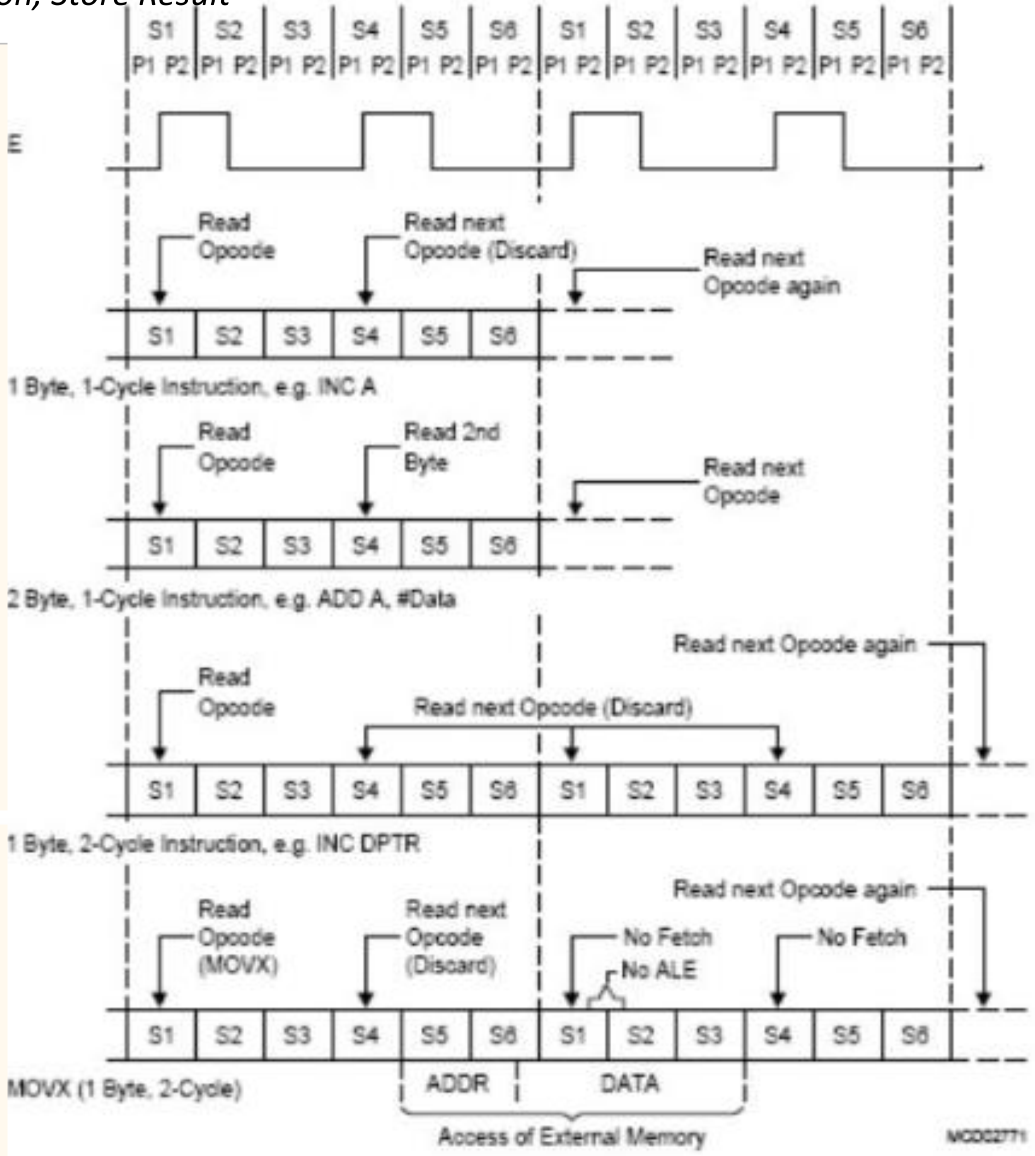


Executing the Instruction:

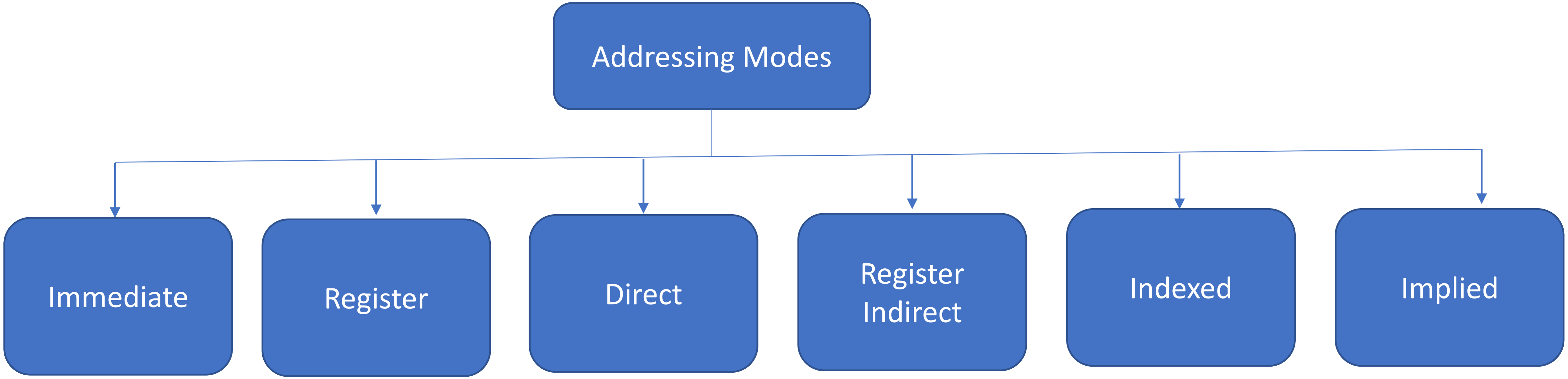
Perform the function of the instruction. If arithmetic or logic instruction, utilize the ALU circuits to carry out the operation on data in registers. This is the only stage of the instruction cycle that is useful from the perspective of the end user. Everything else is overhead required to make the execute stage happen.

Storing the Result:

Store result in memory if necessary: If destination is a memory address, initiate a memory write cycle to transfer the result from the CPU to memory. Depending on the situation, the CPU may or may not have to wait until this operation completes. If the next instruction does not need to access the memory chip where the result is stored, it can proceed with the next instruction while the memory unit is carrying out the write operation.


S1 S2 S3 S4 S5 S6 S1 S2 S3 S4 S5 S6
P1 P2
E
Read Opcode
Read next Opcode (Discard)
Read next Opcode again
1 Byte, 1-Cycle Instruction, e.g. INC A
Read Opcode
Read 2nd Byte
Read next Opcode
2 Byte, 1-Cycle Instruction, e.g. ADD A, #Data
Read next Opcode again
Read Opcode
Read next Opcode (Discard)
1 Byte, 2-Cycle Instruction, e.g. INC DPTR
Read next Opcode again
Read Opcode (MOVX)
Read next Opcode (Discard)
No Fetch
No ALE
No Fetch
MOVX (1 Byte, 2-Cycle)
ADDR
DATA
Access of External Memory
MCD02771

8051 Addressing Modes



Addressing Modes

Immediate

Data is provided in the instruction itself, immediately after the opcode
Ex. **MOVA, #0AFH**
Note: # symbol is used for immediate data

Register

Source or destination data should be present in a register (R0 to R7) but not both
Ex. **MOVA, R5**

Direct

Source or destination address is specified by using 8-bit data in the instruction. Only the internal data memory can be used in this mode
Ex. **MOVR2, 45H**

Register Indirect

Source or destination address is given in the register. By using register indirect addressing mode, the internal or external addresses can be accessed. The R0 and R1 are used for 8-bit addresses, and DPTR is used for 16-bit addresses, no other registers can be used for addressing purposes
Ex. **MOV@DPTR, A**
MOV@R1, 80H

Indexed

Source memory can only be accessed from program memory only. The destination operand is always the register A
Ex. **MOVCA, @A+PC**

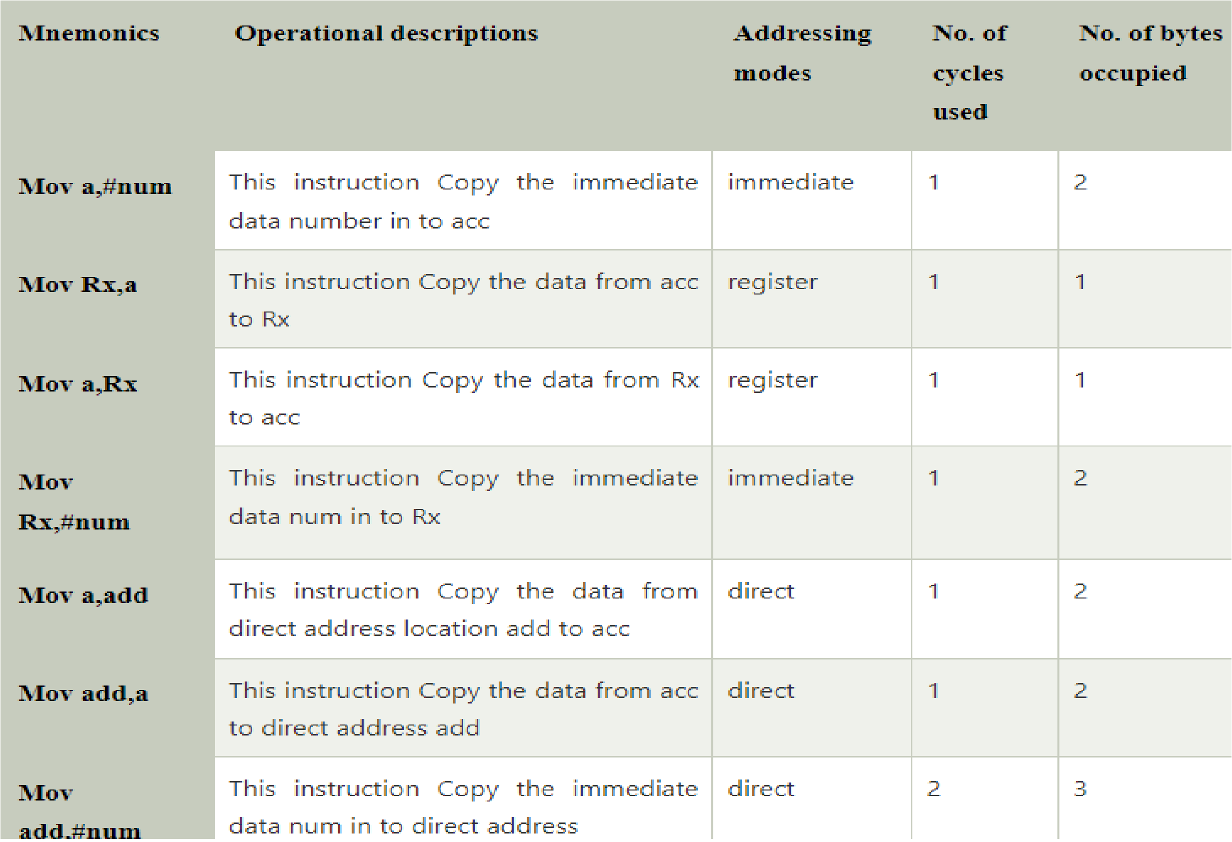
Implied

There will be a single operand. These types of instruction can work on specific registers only. 1-byte instruction
Ex. **RLA**

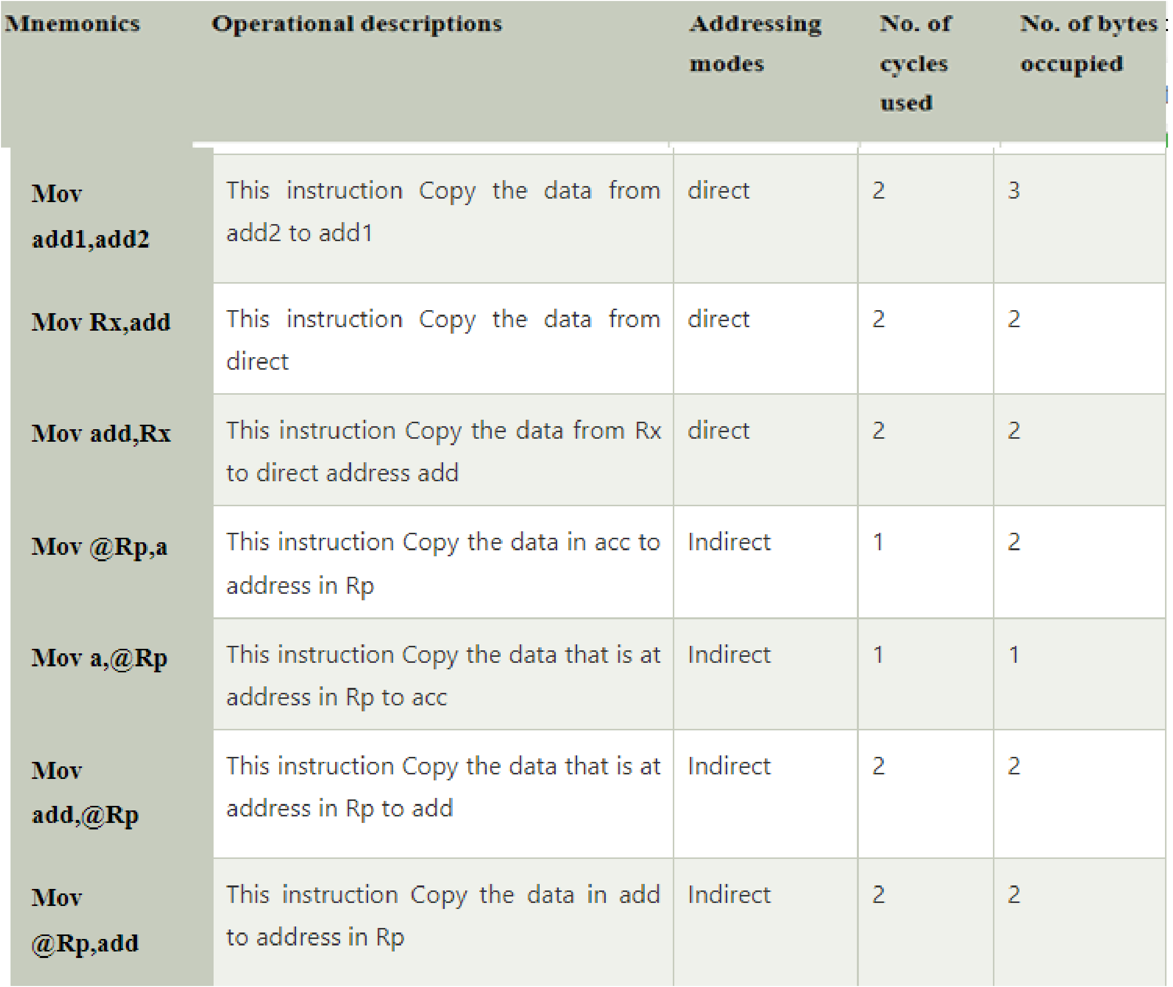
8051 Instruction Set

Collection of instructions that the microcontroller is designed to execute.

DATA MOVING OR HANDLING INSTRUCTIONS

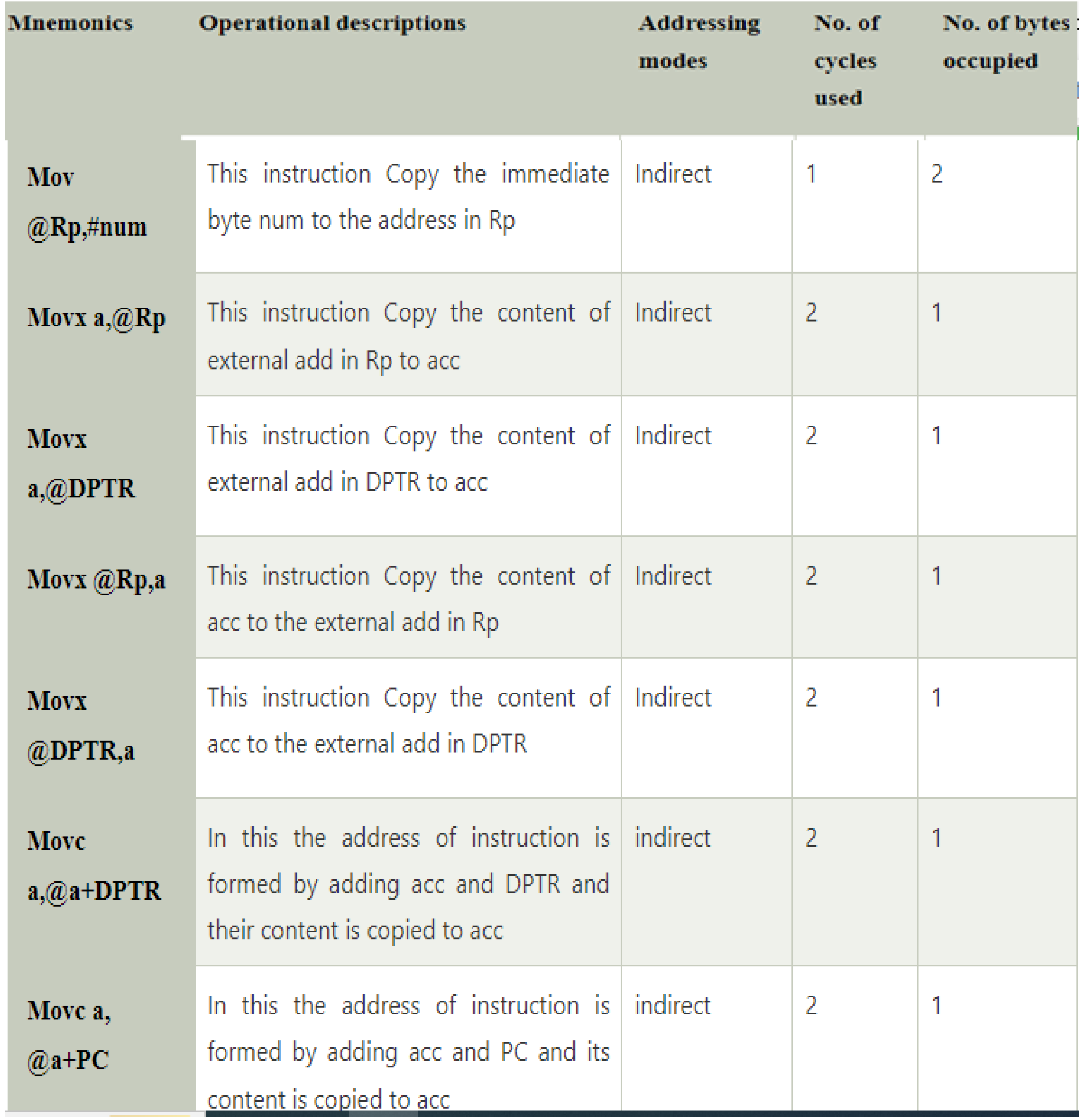


Mnemonics	Operational descriptions	Addressing modes	No. of cycles used	No. of bytes occupied
Mov a,#num	This instruction Copy the immediate data number in to acc	immediate	1	2
Mov Rx,a	This instruction Copy the data from acc to Rx	register	1	1
Mov a,Rx	This instruction Copy the data from Rx to acc	register	1	1
Mov Rx,#num	This instruction Copy the immediate data num in to Rx	immediate	1	2
Mov a,add	This instruction Copy the data from direct address location add to acc	direct	1	2
Mov add,a	This instruction Copy the data from acc to direct address add	direct	1	2
Mov add,#num	This instruction Copy the immediate data num in to direct address	direct	2	3

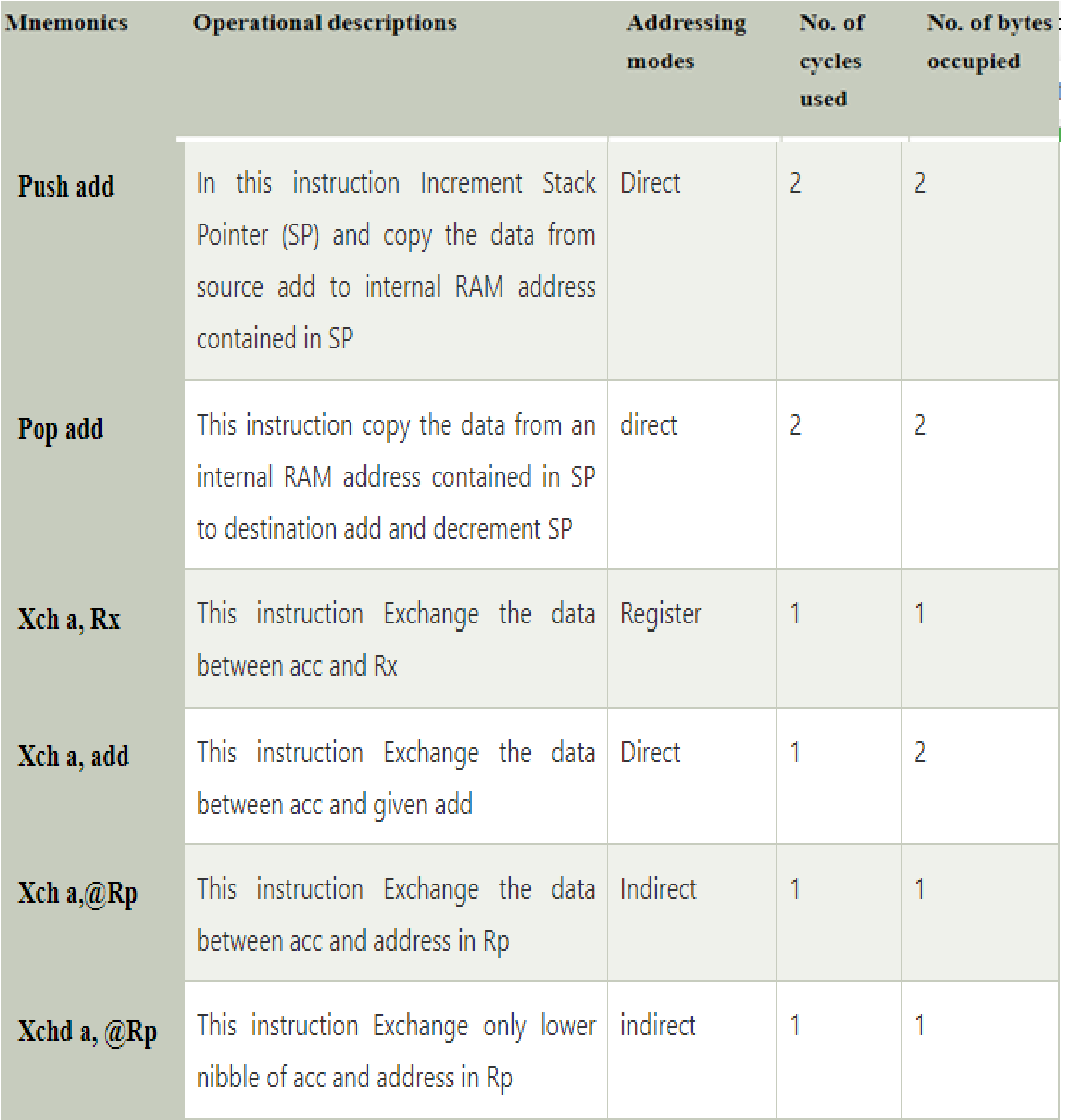


Mnemonics	Operational descriptions	Addressing modes	No. of cycles used	No. of bytes occupied
Mov add1,add2	This instruction Copy the data from add2 to add1	direct	2	3
Mov Rx,add	This instruction Copy the data from direct	direct	2	2
Mov add,Rx	This instruction Copy the data from Rx to direct address add	direct	2	2
Mov @Rp,a	This instruction Copy the data in acc to address in Rp	Indirect	1	2
Mov a,@Rp	This instruction Copy the data that is at address in Rp to acc	Indirect	1	1
Mov add,@Rp	This instruction Copy the data that is at address in Rp to add	Indirect	2	2
Mov @Rp,add	This instruction Copy the data in add to address in Rp	Indirect	2	2

DATA MOVING OR HANDLING INSTRUCTIONS



Mnemonics	Operational descriptions	Addressing modes	No. of cycles used	No. of bytes occupied
Mov @Rp,#num	This instruction Copy the immediate byte num to the address in Rp	Indirect	1	2
Movx a,@Rp	This instruction Copy the content of external add in Rp to acc	Indirect	2	1
Movx a,@DPTR	This instruction Copy the content of external add in DPTR to acc	Indirect	2	1
Movx @Rp,a	This instruction Copy the content of acc to the external add in Rp	Indirect	2	1
Movx @DPTR,a	This instruction Copy the content of acc to the external add in DPTR	Indirect	2	1
Movc a,@a+DPTR	In this the address of instruction is formed by adding acc and DPTR and their content is copied to acc	indirect	2	1
Movc a, @a+PC	In this the address of instruction is formed by adding acc and PC and its content is copied to acc	indirect	2	1



Mnemonics	Operational descriptions	Addressing modes	No. of cycles used	No. of bytes occupied
Push add	In this instruction Increment Stack Pointer (SP) and copy the data from source add to internal RAM address contained in SP	Direct	2	2
Pop add	This instruction copy the data from an internal RAM address contained in SP to destination add and decrement SP	direct	2	2
Xch a, Rx	This instruction Exchange the data between acc and Rx	Register	1	1
Xch a, add	This instruction Exchange the data between acc and given add	Direct	1	2
Xch a,@Rp	This instruction Exchange the data between acc and address in Rp	Indirect	1	1
Xchd a, @Rp	This instruction Exchange only lower nibble of acc and address in Rp	indirect	1	1

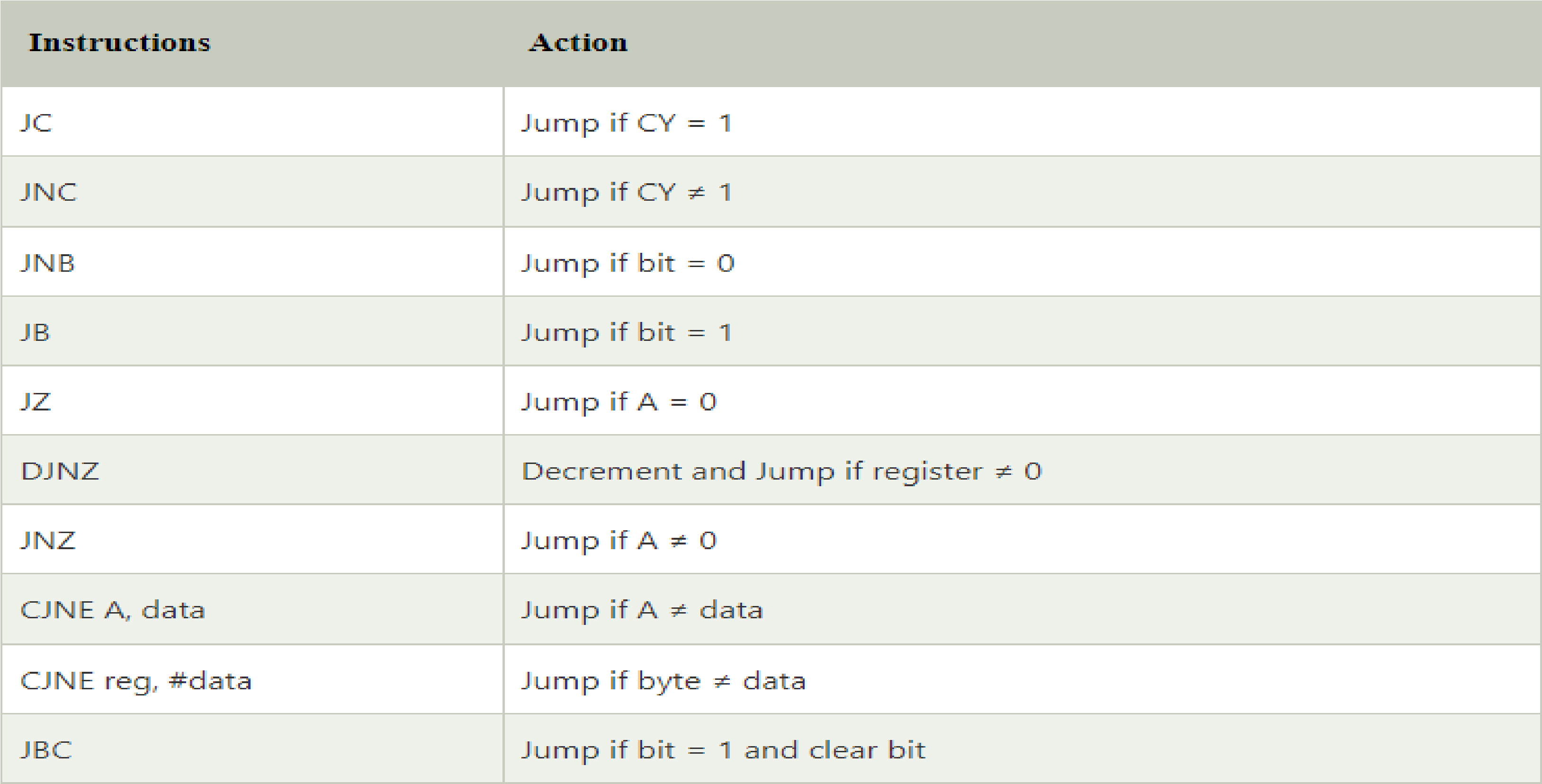
LOOP AND JUMP INSTRUCTIONS

Looping operation in the 8051:

- On repeating a sequence of instructions a certain number of times will result in the formation of loop. The looping operation is used for running the same set of subroutine inside a program number of times as per the requirement.

- Consider the instruction **DJNZ register; label** is used for performing a loop operation. In this instruction, the register is decremented by 1; if this is not zero, then 8051 jumps to the target address referred by the label

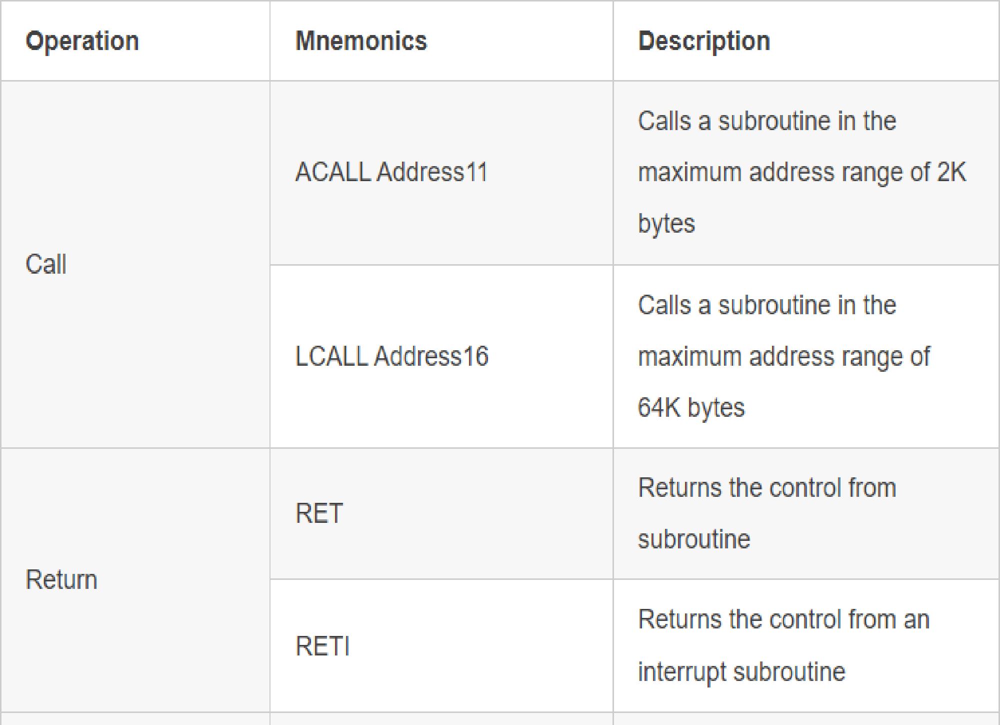
Conditional Jump Instruction:



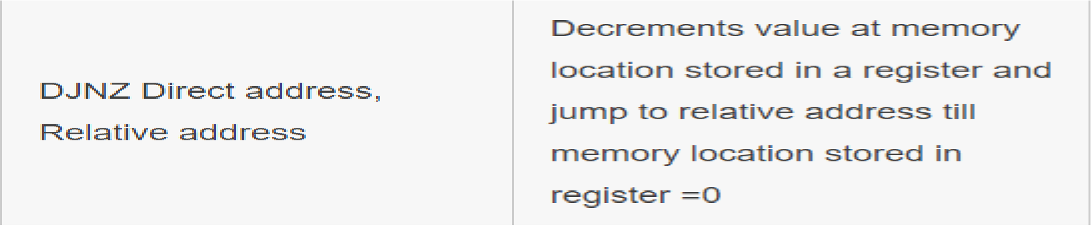
Instructions	Action
JC	Jump if CY = 1
JNC	Jump if CY ≠ 1
JNB	Jump if bit = 0
JB	Jump if bit = 1
JZ	Jump if A = 0
DJNZ	Decrement and Jump if register ≠ 0
JNZ	Jump if A ≠ 0
CJNE A, data	Jump if A ≠ data
CJNE reg, #data	Jump if byte ≠ data
JBC	Jump if bit = 1 and clear bit

Branching Instructions in 8051

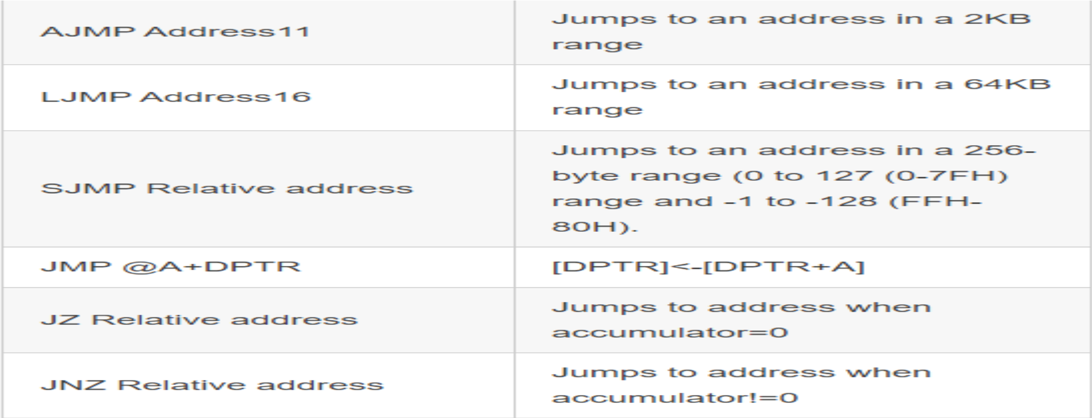
- A microcontroller sequentially executes instructions but in some cases, transferring this control to another block of code becomes essential
- The branching instructions in the 8051 microcontroller are responsible for performing this operation.
- Tasks like looping, **calling delays, and conditional execution** of code can be performed using these branching instructions.



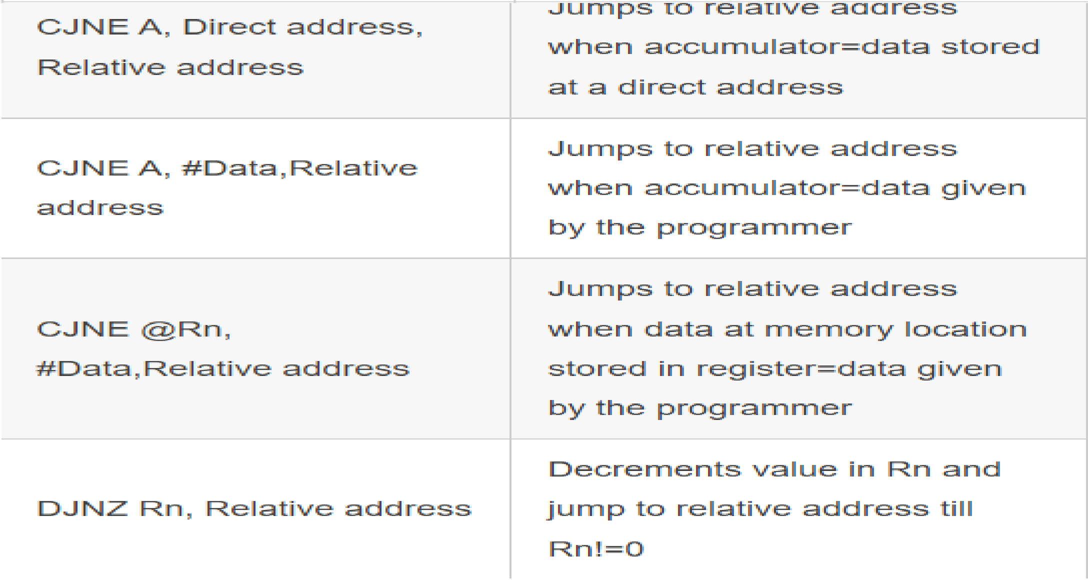
Operation	Mnemonics	Description
Call	ACALL Address11	Calls a subroutine in the maximum address range of 2K bytes
	LCALL Address16	Calls a subroutine in the maximum address range of 64K bytes
Return	RET	Returns the control from subroutine
	RETI	Returns the control from an interrupt subroutine



DJNZ Direct address, Relative address	Decrements value at memory location stored in a register and jump to relative address till memory location stored in register =0



AJMP Address11	Jumps to an address in a 2KB range
LJMP Address16	Jumps to an address in a 64KB range
SJMP Relative address	Jumps to an address in a 256-byte range (0 to 127 (0-7FH) range and -1 to -128 (FFH-80H).
JMP @A+DPTR	[DPTR]<-[DPTR+A]
JZ Relative address	Jumps to address when accumulator=0
JNZ Relative address	Jumps to address when accumulator!=0



CJNE A, Direct address, Relative address	Jumps to relative address when accumulator=data stored at a direct address
CJNE A, #Data,Relative address	Jumps to relative address when accumulator=data given by the programmer
CJNE @Rn, #Data,Relative address	Jumps to relative address when data at memory location stored in register=data given by the programmer
DJNZ Rn, Relative address	Decrements value in Rn and jump to relative address till Rn!=0

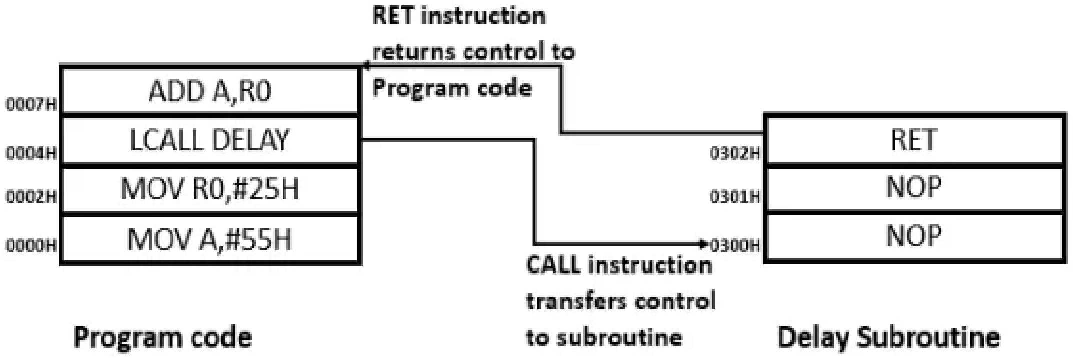
Branching Instructions in 8051

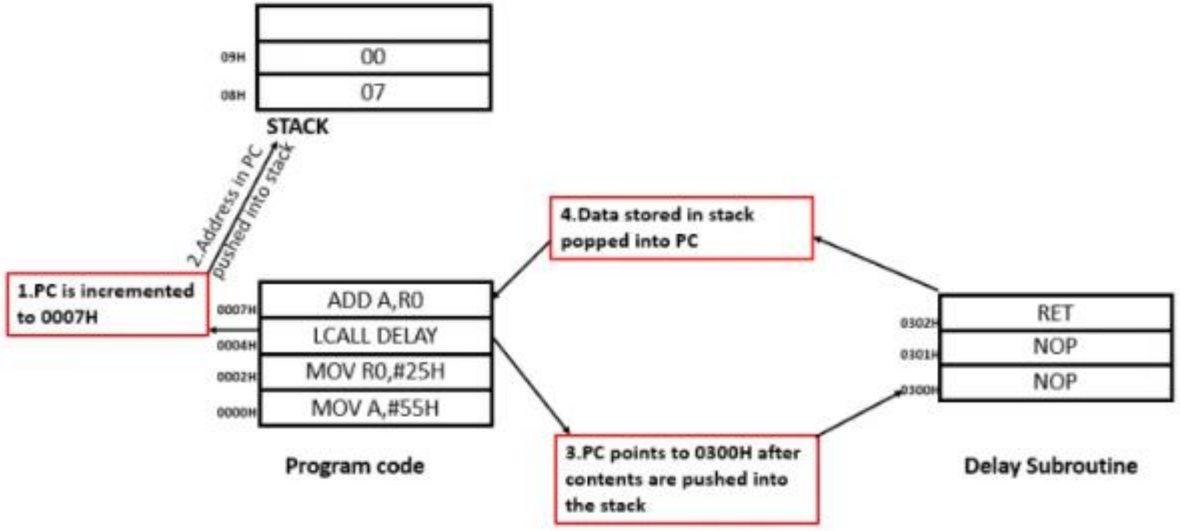
Call and Return instructions in 8051

- A microcontroller needs to perform the same tasks multiple numbers of times across the program, such as generating a delay. A subroutine is responsible for performing these repetitive tasks. Using subroutines saves memory and makes the program more efficient.
- Call instruction is used to transfer the control from the presently executing program code to the subroutine, and the return instruction is used to return the control to the program code

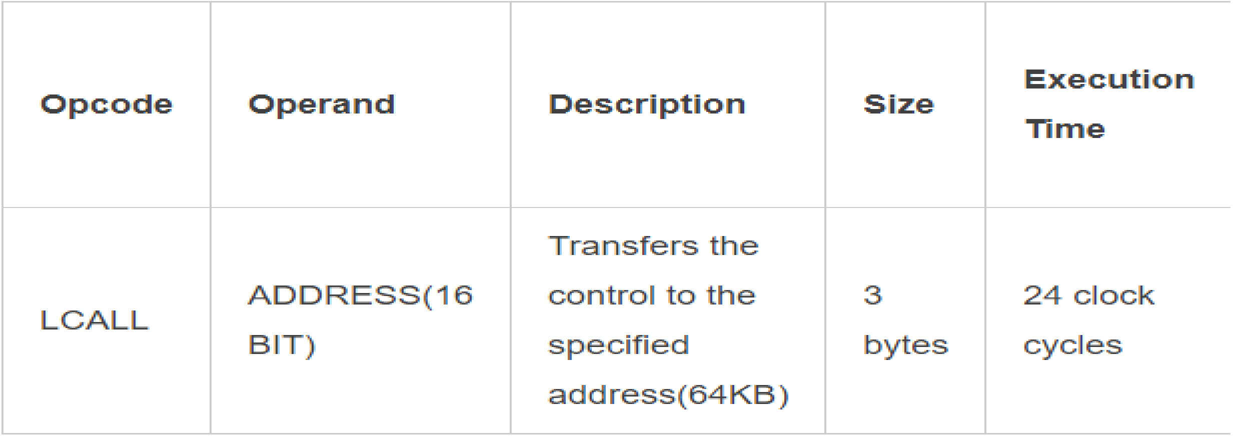
Stack and control transfer

- The stack of the 8051 plays a crucial role in the transfer of control when it comes to call instructions.
- When the call instruction executes, the program counter increments so that it points to the next instruction. The contents of the program counter are pushed into the stack (lower byte first).
- After this, the program counter is loaded with the starting instruction of the subroutine transferring the control
- Once the RET instruction is encountered, the contents stored in the stack are popped back into the program counter, and the microcontroller starts executing the program code from where it had left it




09H
00
08H
07
STACK
2.Address in PC pushed into stack
1.PC is incremented to 0007H
4.Data stored in stack popped into PC
0007H
ADD A,R0
0004H
LCALL DELAY
0002H
MOV R0,#25H
0000H
MOV A,#55H
0302H
RET
0301H
NOP
0300H
NOP
Program code
3.PC points to 0300H after contents are pushed into the stack
Delay Subroutine

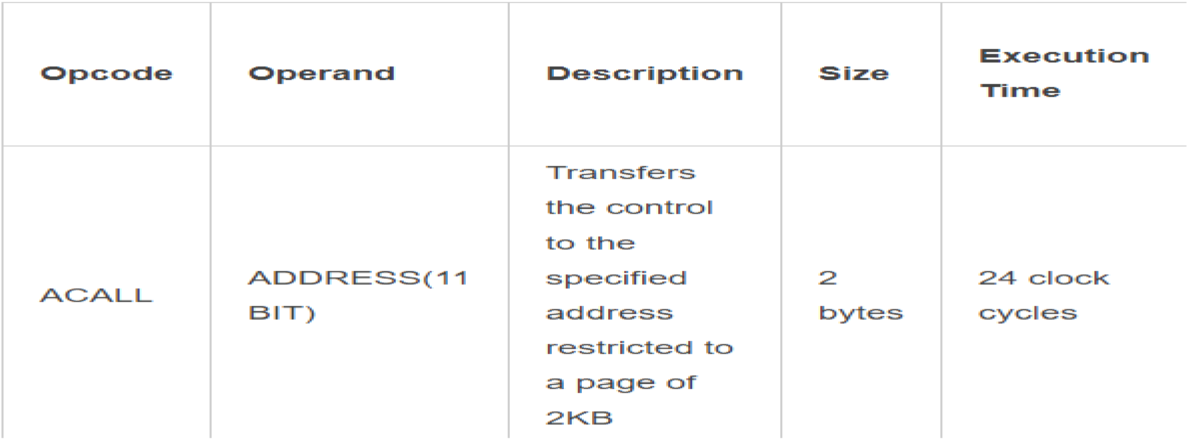
LCALL Instruction: NO FLAGS AFFECTED



Opcode	Operand	Description	Size	Execution Time
LCALL	ADDRESS(16 BIT)	Transfers the control to the specified address(64KB)	3 bytes	24 clock cycles

LCALL instruction can access an address of 16 bits. Due to this reason, this instruction can access any memory location in the ROM space, but this instruction takes up 3 bytes of space and can waste memory resources. To save memory, the ACALL instruction is used.

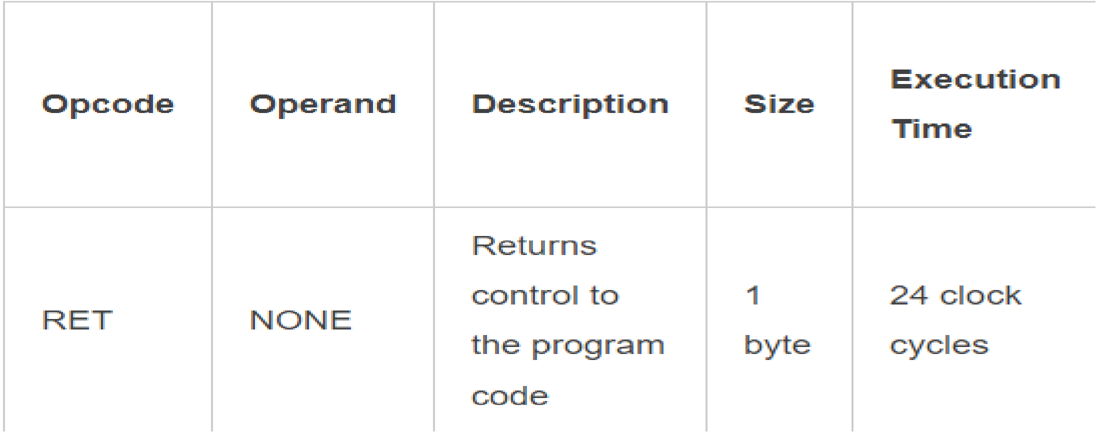
ACALL Instruction: NO FLAGS AFFECTED



Opcode	Operand	Description	Size	Execution Time
ACALL	ADDRESS(11 BIT)	Transfers the control to the specified address restricted to a page of 2KB	2 bytes	24 clock cycles

The ACALL instruction is 2 bytes in size and can be used to access any address in a 2KB page. This instruction affects only 11 bits or the program counter as compared to 16 in the case of LCALL

RET Instruction: *NO FLAGS AFFECTED*



Opcode	Operand	Description	Size	Execution Time
RET	NONE	Returns control to the program code	1 byte	24 clock cycles

The RET instruction is used to return from interrupt subroutines and works in the same way as the RET instruction if used outside a subroutine. When used inside a subroutine, RETI first enables interrupts of equal and lower priorities to the interrupt that it is used in. The program execution continues at the address that is calculated by popping the topmost 2 bytes off the stack. The most-significant-byte is popped off the stack first, followed by the least-significant-byte

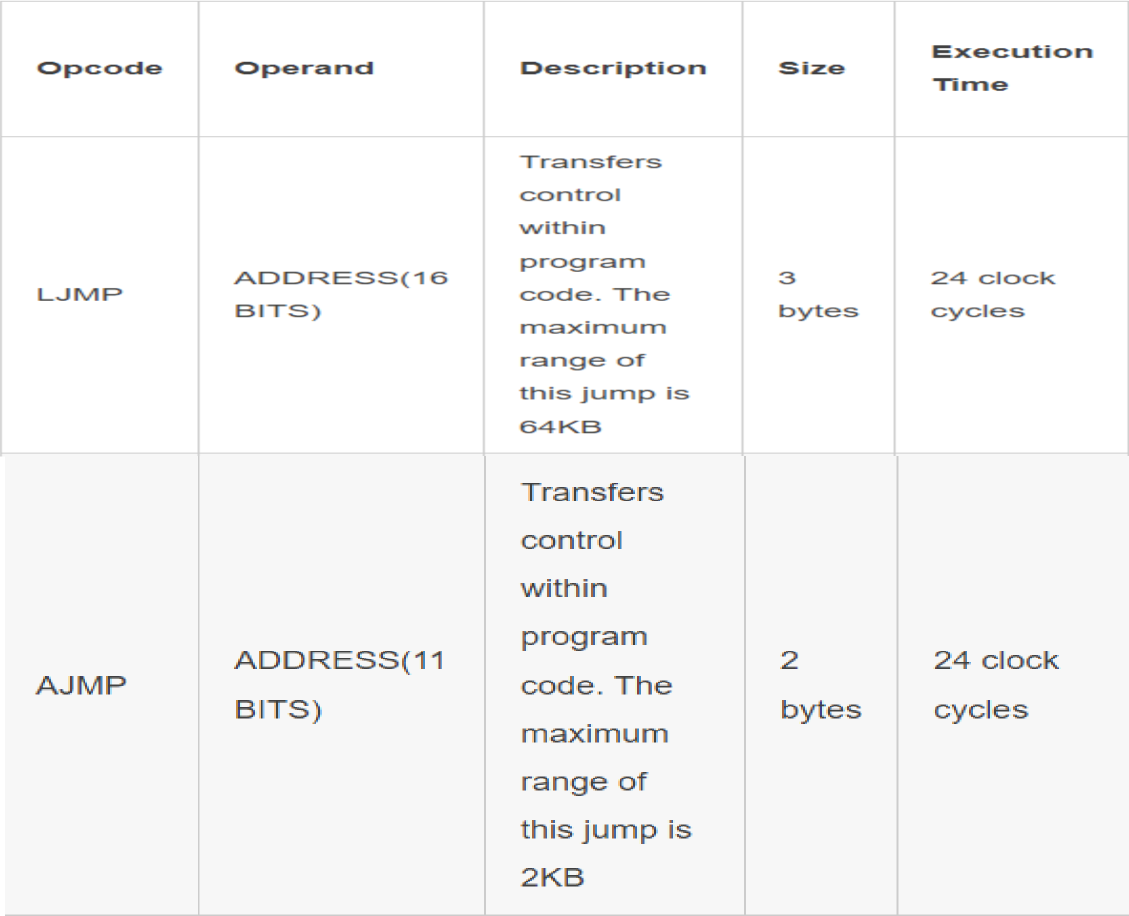
Jumps in 8051

Jump instruction is also used to transfer control in the 8051 microcontroller. But unlike a Call instruction, it does not call a subroutine and jumps to an address in the same program memory. Jumps in the 8051 microcontroller are used to perform looping and conditional execution of program code

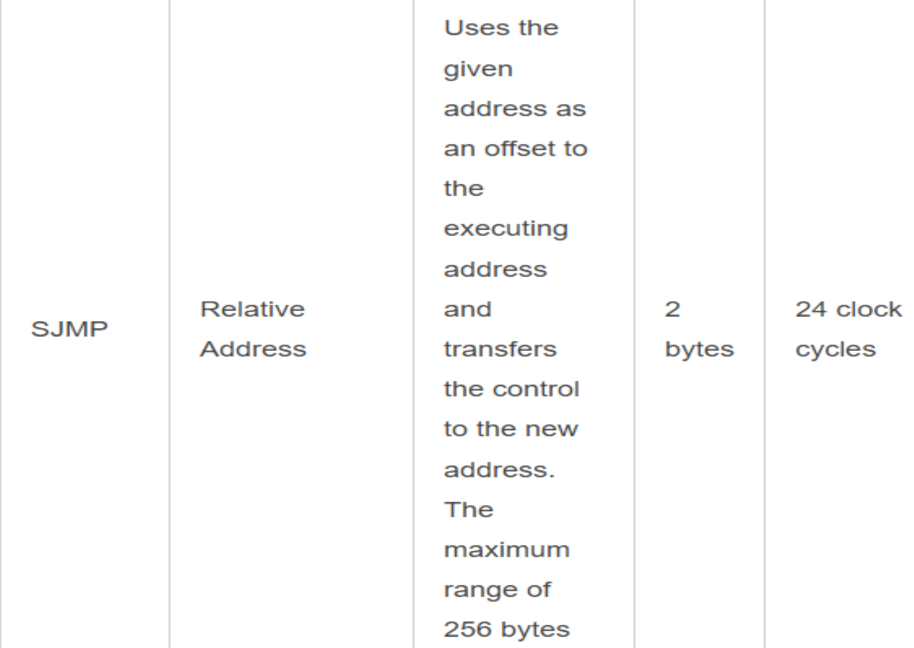
1.Unconditional jumps- These jumps do not evaluate a condition to transfer the control to another address.

2.Conditional jumps- These jumps evaluate a particular condition to transfer the control to another address in the program code. All conditional jumps are short jumps

Unconditional Jump



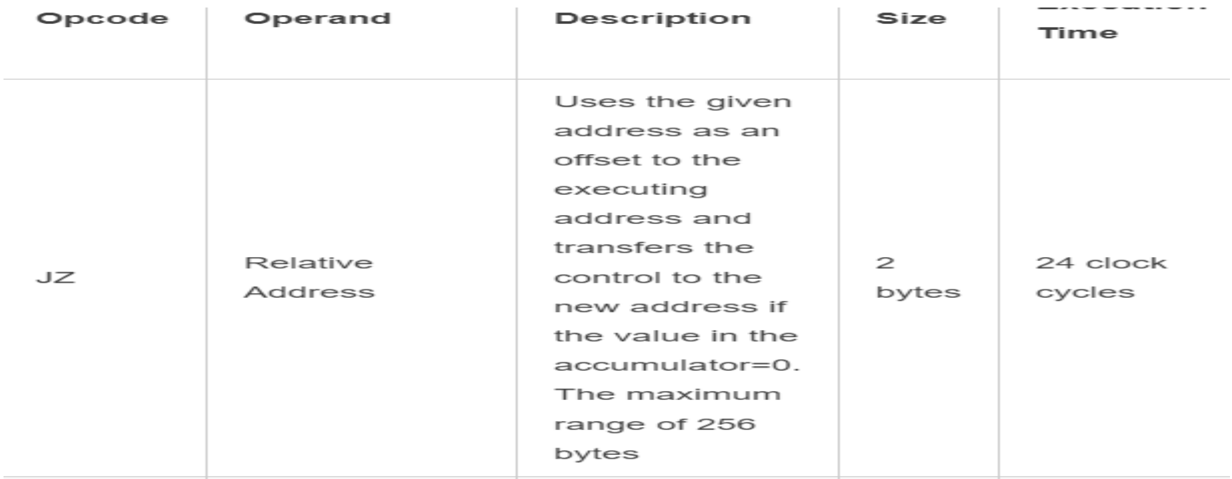
Opcode	Operand	Description	Size	Execution Time
LJMP	ADDRESS(16 BITS)	Transfers control within program code. The maximum range of this jump is 64KB	3 bytes	24 clock cycles
AJMP	ADDRESS(11 BITS)	Transfers control within program code. The maximum range of this jump is 2KB	2 bytes	24 clock cycles

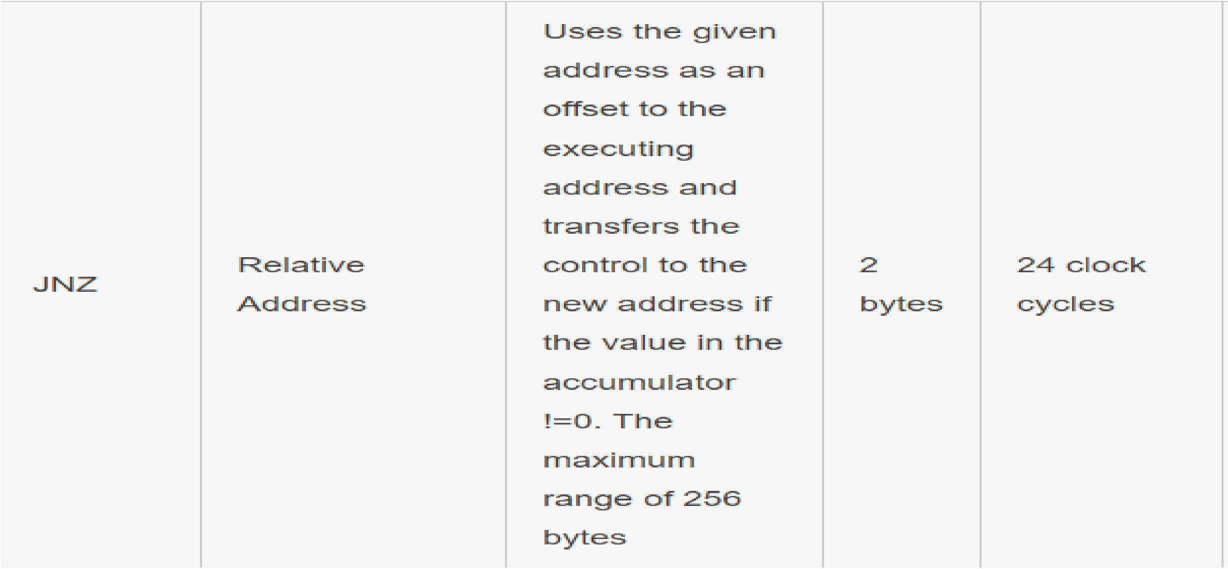


SJMP	Relative Address	Uses the given address as an offset to the executing address and transfers the control to the new address. The maximum range of 256 bytes	2 bytes	24 clock cycles

NO FLAGS AFFECTED

Conditional Jump

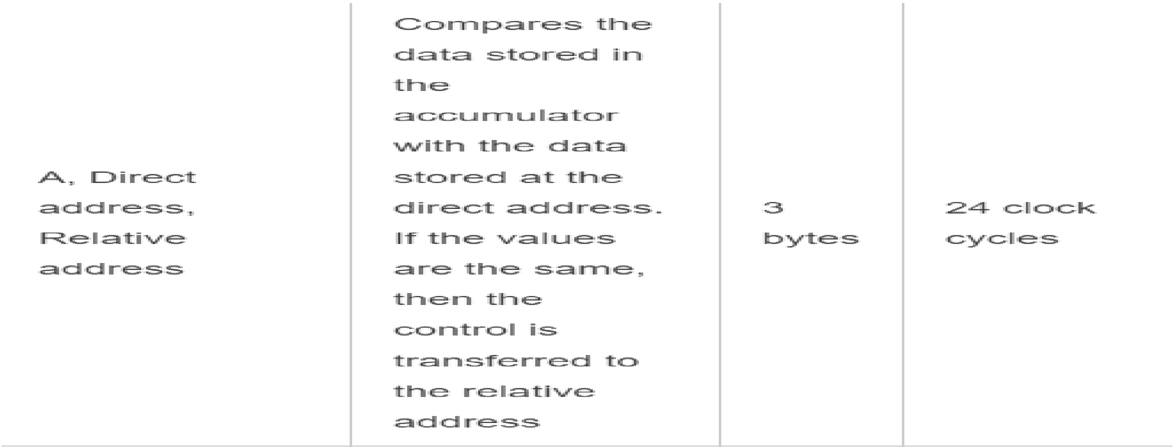


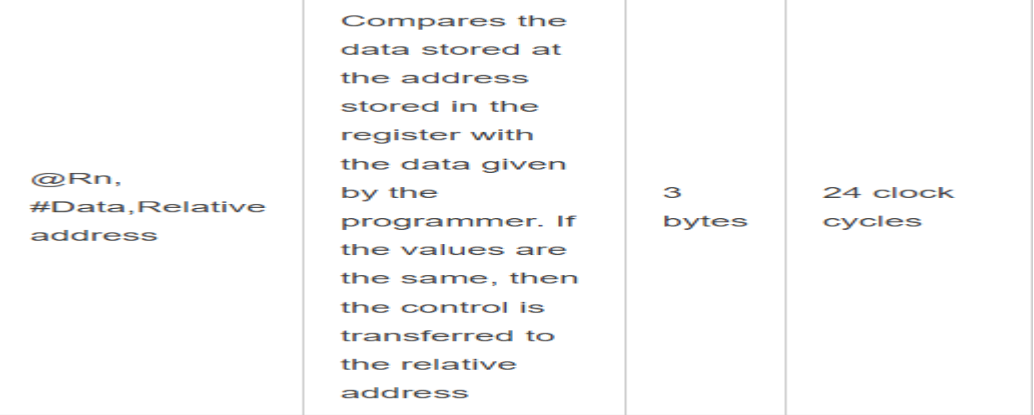


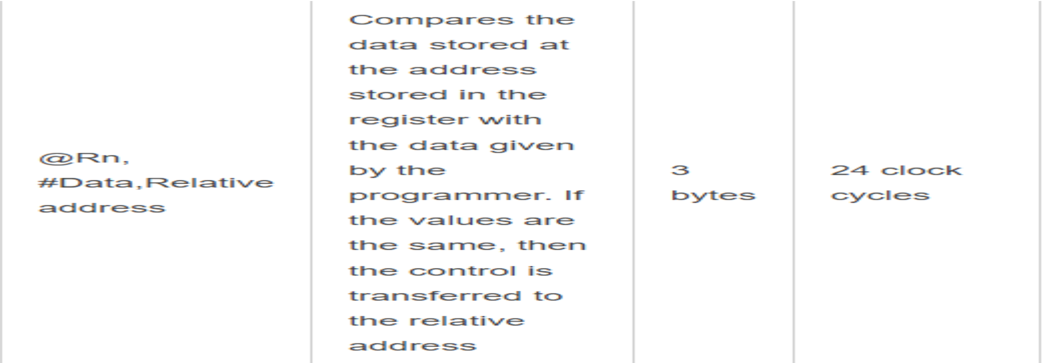
Opcode	Operand	Description	Size	Execution Time
JZ	Relative Address	Uses the given address as an offset to the executing address and transfers the control to the new address if the value in the accumulator=0. The maximum range of 256 bytes	2 bytes	24 clock cycles
JNZ	Relative Address	Uses the given address as an offset to the executing address and transfers the control to the new address if the value in the accumulator !=0. The maximum range of 256 bytes	2 bytes	24 clock cycles

NO FLAGS AFFECTED

CJNE

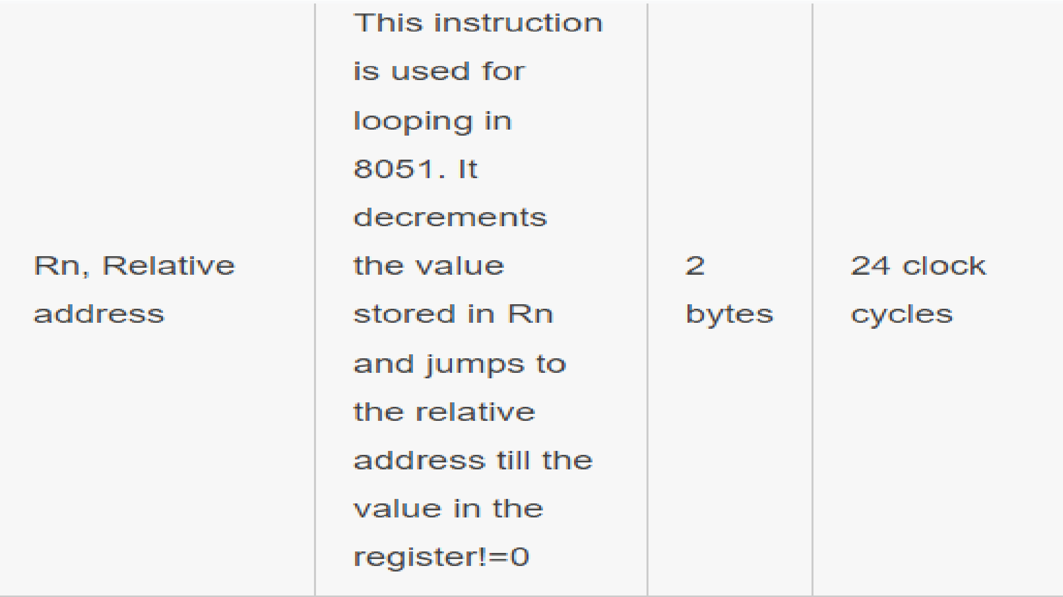




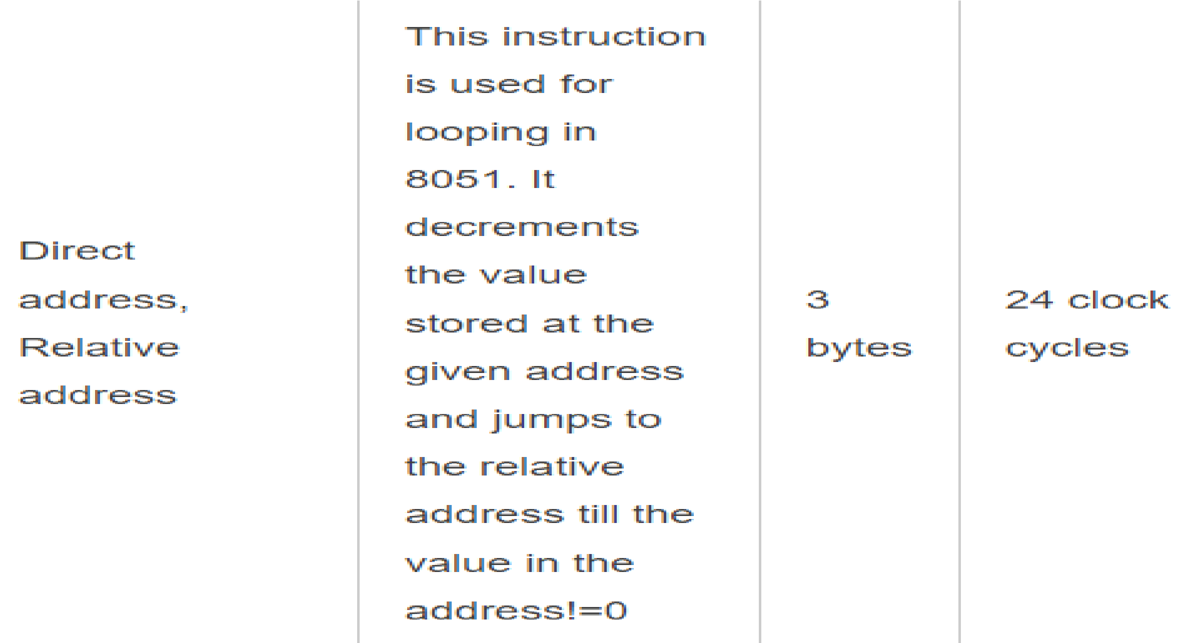


A, Direct address, Relative address	Compares the data stored in the accumulator with the data stored at the direct address. If the values are the same, then the control is transferred to the relative address	3 bytes	24 clock cycles
@Rn, #Data,Relative address	Compares the data stored at the address stored in the register with the data given by the programmer. If the values are the same, then the control is transferred to the relative address	3 bytes	24 clock cycles
@Rn, #Data,Relative address	Compares the data stored at the address stored in the register with the data given by the programmer. If the values are the same, then the control is transferred to the relative address	3 bytes	24 clock cycles

DJNZ



Rn, Relative address	This instruction is used for looping in 8051. It decrements the value stored in Rn and jumps to the relative address till the value in the register!=0	2 bytes	24 clock cycles



Direct address, Relative address	This instruction is used for looping in 8051. It decrements the value stored at the given address and jumps to the relative address till the value in the address!=0	3 bytes	24 clock cycles

NO FLAGS AFFECTED