# CHAPTER 1

# Architecture of 8085

**Topics - Classification of Instructions, Instruction set of 8085, Introduction to 8051 Microcontrollers**

**Text Book Used** – Microprocessors and Microcontrollers- N. Senthi Kumar, M, Saravanam and S Jeevananthan (Oxford University Press)

## 1.1 Classification of Instructions –

Microprocessor instructions can be classified based on parameters such as functionality, length, and operand addressing.

### Based on Functionality

Based on the functionality, the instructions are classified into the following five categories:

1) Data transfer (copy) operations
2) Arithmetic operations
3) Logical operations
4) Branching operations
5) Machine control operations

1) Data transfer (Copy) Operations -

This group of instructions copies data from a location called source register to another location called destination register. Generally, the contents of the source register are not modified. Although the term data transfer is used for the copy operation, it is misleading because it implies that the contents of the source memory location are destroyed. The various types of data transfer are listed in Table 1.3 along with examples of each type.

**Table 1.3** Types of data transfer

| Type | Example |
|---|---|
| Transferring data between one register and another | MOV A, D—Copies the content of register D to the accumulator |
| Storing a data byte in a register or memory location | MVI C, 66H—Loads register C with the data 66H |
| Transferring data between a memory location and a register | LDA 8800H—Loads the contents of memory location 8800H in the accumulator |
| Transferring data between an I/O device and the accumulator | IN PORT1—Transfers data from an input device to the accumulator |

2) Arithmetic Operations -

Arithmetic operations include addition, subtraction, increment, and decrement. As the 8085 has an accumulator-oriented ALU, one of the data used in the arithmetic operations is stored in the accumulator; the result is also stored in the accumulator. Arithmetic and logical operations cannot be executed without the accumulator.

Addition (ADD) -The addition instructions of the 8085 add the contents of a register or memory location with the contents of the accumulator. The result is stored in the accumulator. The Intel 8085 instruction set supports two types of addition instructions—with and without addition of the carry flag content to the least significant bit of the numbers. The instruction set also supports 16-bit addition, i.e., the content of the HL register pair can be added to that of another register pair and the result stored in the HL register pair.

Subtraction (SUB) - The instruction set of the 8085 supports two types of subtraction—with borrow and without borrow. Like addition, the subtraction operation also uses the accumulator as reference, i.e., it subtracts the content of a register or memory location from that of the accumulator and stores the result in the accumulator.

Increment/Decrement - These operations can be used to increment or decrement the contents of any register, register pair, or memory location. Unlike the arithmetic and logical operations, the increment and decrement operations need not be based upon the accumulator.

3) Logical Operations -

Logical instructions are also accumulator-oriented, i.e., they require one of the operands to be placed in the accumulator. The other operand can be any register or memory location. The result is stored in the accumulator. The operations that use two operands are logical AND, OR, and EXOR. The operation that uses a single operand (i.e., the accumulator) is the logical complement or NOT operation.

The instruction set of the 8085 supports rotation of the data stored in accumulator.

The data can be rotated left or right, through the carry or without the carry.

The most important 8085 instruction is the compare instruction. This instruction is used to compare register or memory content with the accumulator content. The result of comparison such as equal to, greater than, or less than is reflected in the flag register bits.

4) Branching Operations-

Branching instructions are important for programming a microprocessor. These instructions can transfer control of execution from one memory location to another, either conditionally or unconditionally. Branching can take place in the following two ways:

Execution control cannot return to the point of branching. Example: Jump instructions

Execution control can return to the point of branching, which is stored by the 8085. Example: Subroutine call instructions

5) Machine Control Operations -

These instructions can be used to control the execution of other instructions. They include halting the operation of the microprocessor, interrupting program execution, etc. Detailed explanations for 8085 instructions are given in Section 1.14.

**Based on Length**

Based on the length of the machine language code, 8085 instructions can be classified into the following three types:

1) One-byte instructions
2) Two-byte instructions
3) Three-byte instructions

Assembly language instructions should be converted into machine code for storage and execution by the processor. So the length of the machine language code instructions determines the length of the program. This in turn determines the amount of memory required for the program.

1) One-byte Instructions

Instructions that require only one byte in machine language are called one-byte instructions. These instructions just have the machine code or opcode alone to represent the operation to be performed. The common examples are the instructions that have their operands within the processor itself. Some examples of one-byte instructions are given in Table 1.4. Even though the instruction ADD M adds the content of a memory location to that of the accumulator, its machine code requires only one byte.

**Table 1.4** One-byte instructions

| Opcode | Operand | Machine code/Opcode/ Hex code |
|---|---|---|
| MOV | A, B | 78 |
| ADD | M | 86 |
| XRA | A | AF |

Let us now understand the instruction MOV Rd, Rs. This instruction copies the contents of source register Rs to destination register Rd. (Rd ⇦ Rs)

It is coded as 01dddsss. Here, ddd is the binary code of one of the seven general-purpose registers that is the destination of the data and sss is the binary code of the source register.

Example: MOV A, B (coded as 01111000 = 78H)

2) Two-byte Instructions

Instructions that require two bytes in machine code are called as two-byte instructions. The first byte of the two-byte instructions is the opcode, which specifies the operation to be performed. The second byte is the 8-bit operand, which is either an 8-bit number or an address. Some common examples of two- byte instructions are listed in Table 1.5.

**Table 1.5** Two-byte instructions

| Opcode | Operand | Machine code/Opcode/Hex code | Byte description |
|---|---|---|---|
| MVI | A, 7FH | 3E | First byte |
| | | 7F | Second byte |
| ADI | 0FH | C6 | First byte |
| | | 0F | Second byte |
| IN | 40H | DB | First byte |
| | | 40 | Second byte |

The instruction is stored in two consecutive memory locations.

MVI R, data (R ← data)

*Example:*

MVI A, 32H (coded as 3E 32 in two contiguous bytes) This is an example of immediate addressing.

3) Three-byte Instructions

Instructions that require three bytes in machine code are called three-byte instructions. In 8085 machine language, the first byte of the three-byte instructions is the opcode which specifies the operation to be performed. The next two bytes refer to the 16-bit operand, which is either a 16-bit number or the address of a memory location. Some common examples of three-byte instructions are listed in Table 1.6.

**Table 1.6** Three-byte instructions

| Opcode | Operand | Machine code/Opcode/Hex code | Byte description |
|---|---|---|---|
| JMP | 9050H | C3 | First byte |
| | | 50 | Second byte |
| | | 90 | Third byte |
| LDA | 8850H | 3A | First byte |
| | | 50 | Second byte |
| | | 88 | Third byte |
| LXI | H, 0520H | 21 | First byte |
| | | 20 | Second byte |
| | | 05 | Third byte |

The instruction LXI Rp, 16-bit data can be explained as follows:
Rp is one of the pairs of registers BC, DE, or HL, which are used as 16-bit registers. The two data bytes are to be stored as a 16-bit number in L and H in sequence. LXI H, 0520H is coded as 21H 20H 05H in three bytes. (This is an example of immediate addressing.)
In executing the instruction LDA addr, the accumulator is loaded with the memory content of the address given in the instruction. Addr is a 16-bit address. LDA 8850H is coded as 3AH 50H 88H. (This is an example of direct addressing.)

**Addressing Modes in Instructions**
Every instruction in a program has to operate on data. The process of specifying the data to be operated on by the instruction is called *addressing*. Efficient software development for the microprocessor requires complete familiarity with the addressing mode employed for each instruction. For example, the instructions MOV B, A and MVI A, 82H are used to copy data from a source to a destination. In these instructions, the source can be a register or an 8-bit number (00H to FFH); the destination is a register. The source and destination are operands. The various formats for specifying operands are called *addressing modes*. The 8085 has the following five types of addressing:

1) Immediate addressing
2) Memory direct addressing
3) Register direct addressing
4) Indirect addressing
5) Implied or implicit addressing

1) Immediate Addressing

Immediate addressing transfers the operand given in the instruction—a byte or word—to the destination register or memory location. The operand is part of the instruction. The format for immediate addressing is given in Fig. 1.10.

*Example:* MVI A, 9AH

a) The operand is part of the instruction.
b) The operand is stored in the register mentioned in the instruction.

*Example:* ADI 05H

a) Add 05H to the contents of the accumulator.
b) 05H is the operand.

Immediate addressing has no memory reference to fetch data. It executes faster, but has limited data range.

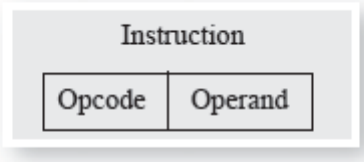



**Fig. 1.10** Format of immediate addressing

2) Memory Direct Addressing

Memory direct addressing moves a byte or word between a memory location and register. The memory location address is given in the instruction. The instruction set does not support memory-to-memory transfer. Memory direct addressing is illustrated in Fig. 1.11.

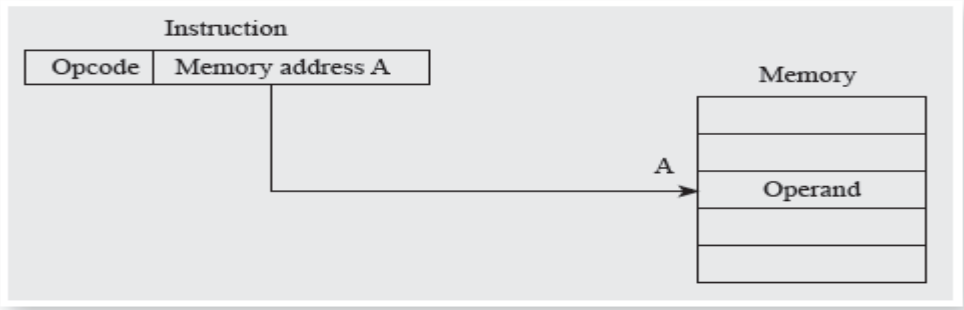



**Fig. 1.11** Format of memory direct addressing

Example: LDA 850FH

This instruction is used to load the contents of the memory location 850FH in the accumulator.

Example: STA 9001H

This instruction is used to store the contents of the accumulator in the memory address 9001H.

In these instructions, the memory address of the operand is given in the instruction. Direct addressing is also used for data transfer between the processor and input/output devices. For example, the IN instruction is used to receive data from the input port and store it in the accumulator; the OUT instruction is used to send the data from the accumulator to the output port.

Example: IN 00H and OUT 01H

3) Register Direct Addressing

Register direct addressing transfers a copy of a byte or word from the source register to the destination register. The operand is in the register named in the instruction. It executes very fast, has very limited register space, and requires good assembly programming. The operand is within in the processor itself; so the execution is faster. Register direct addressing is illustrated in Fig. 1.12.

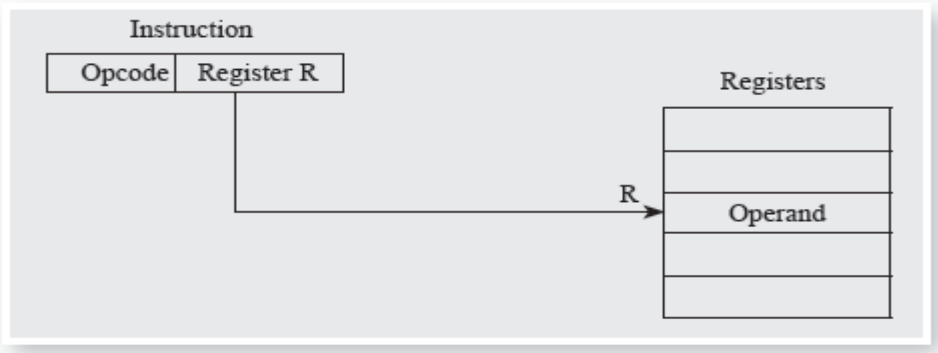



**Fig. 1.12** Format of register direct addressing

*Example:* MOV Rd, Rs

MOV B, C

It copies the contents of register C to register B.

*Example:* ADD B

It adds the contents of register B to the accumulator and saves it in the accumulator.

4) Indirect Addressing

Indirect addressing transfers a byte or word between a register and a memory location. The address of a memory location is stored in a register and that register is specified in the instruction. This is illustrated in Fig. 1.13.

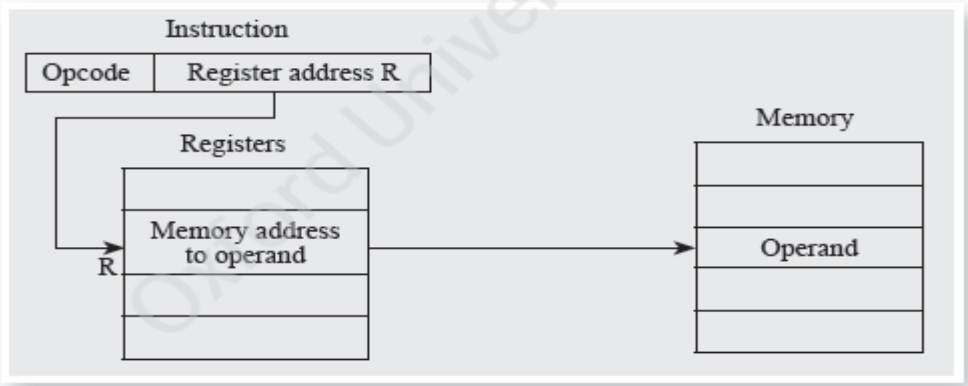



**Fig. 1.13** Format of indirect addressing

In indirect addressing, the effective address is calculated by the processor using the contents of the register specified in the instruction. This type of addressing employs several accesses—two accesses to retrieve the 16-bit address and a further access (or accesses) to retrieve the data which is to be loaded in the register.

*Example:* MOV A, M

Here, the data is in the memory location pointed to by the contents of the HL pair. The data is moved to the accumulator.

5) Implied or Implicit Addressing

In implied addressing mode, the instruction itself specifies the data to be operated upon. For example, CMA complements the contents of the accumulator. No specific data or operand is mentioned in the instruction.

### 1.2 Instruction set of 8085 –

The 8085 microprocessor instruction set has 74 operation codes and 246 instructions. It is compatible with that of its predecessor, the 8080A, but has two additional instructions—SIM (set interrupt mask) and RIM (read interrupt mask)—related to serial I/O.

**Format of Assembly Language Instructions and Programs -** Assembly language programs are written for performing specific functions, converted into machine language code, and then stored in the memory of the microprocessor-based system. The conversion of an assembly language program into machine language code is called assembling; the application that performs this task is called assembler. This conversion or assembling can also be done manually by the programmers. To facilitate the process of assembling, the assembly language programs are written in a specific format as shown in Fig. 1.14.

| Memory address | Machine code/Opcode | Label | Mnemonics with operands | Comments |
|---|---|---|---|---|

**Fig. 1.14** Format for writing assembly language programs

In general, the assembly language mnemonics with their operands are written first. The address where the instructions are stored is given a dummy name called label. The purpose of labels is to give the correct branch addresses in instructions. Labels are separated from mnemonics with a colon.

The comments column is essential for any program as it helps the programmer understand the logic of the program at any point in time. Without comments, it is difficult to understand an assembly language program. Comments are separated from the mnemonics with a semicolon.

The first two columns correspond to the physical memory address and the actual machine code. These two columns are filled in after completing the assembly language programming. These columns must contain only binary numbers, but for easy understanding, hexadecimal numbers are used. For manual assembling, these two columns are filled in by the programmer. An assembler can generate these columns automatically.

An example of the assembly language program format is given in Table 1.7.

**Table 1.7** Sample assembly language program

| Memory address | Machine code/ Opcode | Label | Mnemonics with operands | Comments |
|---|---|---|---|---|
| 8000 | 3E | START: | MVI A, 5FH | ; Load data in the accumulator. |
| 8001 | 5F | | | |
| 8002 | | | | ; Address of the next memory location |

The instruction in Table 1.7 moves the data 5FH to the accumulator.

**Data Transfer Instructions**

Data transfer instructions are used to transfer data between two registers in the microprocessor or between a peripheral device and the microprocessor. Some instructions and their features are given in the following points. The complete list with explanations is given in Table 1.8.

i) MVI instruction is used for storing 8-bit data in a microprocessor register.
ii) LXI instruction is used for storing 16-bit data in a register pair.
iii) In direct addressing mode, MOV instruction is used for data transfer between registers. In indirect addressing mode, MOV is used for data transfer between a memory location and a register. If the instruction has M in the operand field, the memory location pointed to by the HL pair is considered for data transfer.
iv) LDA and STA use memory direct addressing mode and a 16-bit memory address as operand.
v) LDAX and STAX use indirect addressing mode for data transfer. The operand given in the instruction is one of the register pairs BC or DE. Register pair HL is not used with LDAX due to the availability of the alternative instruction MOV A, M.
vi) LHLD and SHLD are the instructions used to transfer 16-bit data between the HL register pair and two consecutive memory locations. For example, executing SHLD 9000H instruction will store the contents of L register in 9000H and the contents of H register in 9001H.
vii) PUSH and POP instructions are used for data transfer between a register pair and a stack. The stack is a set of memory locations configured as a last- in, first-out (LIFO) or first-in, last-out (FILO) array. The top of the stack locations is pointed to by a special register, the stack pointer, which is within the microprocessor. PUSH instruction will store the register pair given in the instruction to the top two memory locations of the stack. Similarly, POP instruction will copy the last two bytes stored in the stack to the register pair mentioned in the instruction. Care must be taken in using these instructions as the stack is configured as a LIFO array. Another instruction to store data in the stack is XTHL, which exchanges the top two memory locations of the stack with the contents of the HL register pair.
viii) Stack pointer can be initialized using LXI or SPHL instructions. SPHL instruction will copy the contents of the HL register pair to the stack pointer.
ix) IN and OUT instructions use 8-bit port addresses as operand. IN instruction is used to get data from the input port and the data obtained is stored in the accumulator. OUT instruction is used to issue data from the accumulator to an output port.
x) XCHG instruction is used to exchange the contents of the HL and DE register pairs.

**Table 1.8** Data transfer instructions

| Mnemonics | Tasks performed on execution | Addressing mode | Instruction length | Example |
|---|---|---|---|---|
| MVI R, 8-bit | Moves the 8-bit data to the register | Immediate | Two bytes | MVI B, 3FH |
| LXI Rp, 16-bit | Loads the 16-bit data in the register pair | Immediate | Three bytes | LXI B, 5AF3H |
| MOV Rd, Rs | Copies the data from the source register to the destination register | Register direct | One byte | MOV A, B |
| LDA 16-bit | Loads the accumulator with the data from the memory location indicated by the 16-bit address | Memory direct | Three bytes | LDA 905FH |
| LHLD 16-bit | Loads the H and L registers directly from the two consecutive memory locations indicated by the 16-bit address | Memory direct | Three bytes | LHLD 900AH |
| STA 16-bit | Stores the contents of the accumulator in the memory location indicated by the 16-bit address | Memory direct | Three bytes | STA 9050H |
| SHLD 16-bit | Stores the contents of the H and L registers in two consecutive memory locations indicated by the 16-bit address | Memory direct | Three bytes | SHLD 809FH |
| PUSH Rp | Pushes the contents of the register pair onto a stack | Register direct | One byte | PUSH B |
| POP Rp | Pops the top two memory locations of the stack onto a register pair | Register direct | One byte | POP H |
| OUT 8-bit | Outputs the data in the accumulator to the port indicated by the 8-bit address | I/O | Two bytes | OUT 40H |
| IN 8-bit | Inputs the data from the port indicated by the 8-bit address to the accumulator | I/O | Two bytes | IN 30H |
| MOV Rd, M | Copies the contents of the memory location pointed to by the HL register pair to the register | Indirect | One byte | MOV B, M |
| MOV M, Rs | Copies the contents of the register to the memory location pointed to by the HL register pair | Indirect | One byte | MOV M, C |
| LDAX Rp | Loads accumulator with the contents of the memory location pointed to by the register pair | Indirect | One byte | LDAX B |
| STAX Rp | Stores the contents of the accumulator in the memory location pointed to by the register pair | Indirect | One byte | STAX D |
| XCHG | Exchanges the contents of the HL register pair with that of the D and E | Implicit | One byte | XCHG |
| SPHL | Copies the contents of the H and L registers to the stack pointer | Implicit | One byte | SPHL |
| XTHL | Exchanges the contents of the HL register pair with the top of stack | Implicit | One byte | XTHL |

**Arithmetic Instructions**

The arithmetic instructions supported by the 8085 are addition, subtraction, and their variants. The arithmetic instructions are listed in Table 1.9.

**Table 1.9** Arithmetic instructions

| Mnemonics | Tasks performed on execution | Addressing mode | Instruction length | Example |
|---|---|---|---|---|
| ADI 8-bit | Adds the 8-bit data to the contents of the accumulator | Immediate | Two bytes | ADI 30H |
| ACI 8-bit | Adds the 8-bit data and the carry flag to the contents of the accumulator | Immediate | Two bytes | ACI 4FH |
| SUI 8-bit | Subtracts the 8-bit data from the contents of the accumulator | Immediate | Two bytes | SUI 2AH |
| SBI 8-bit | Subtracts the 8-bit data and the borrow from the contents of the accumulator | Immediate | Two bytes | SBI 5CH |
| ADD R | Adds the contents of the register to the contents of the accumulator | Register direct | One byte | ADD C |
| ADC R | Adds the contents of the register and the carry to the contents of the accumulator | Register direct | One byte | ADC E |
| SUB R | Subtracts the contents of the register from that of the accumulator | Register direct | One byte | SUB B |
| SBB R | Subtracts the contents of the register and the borrow from that of the accumulator | Register direct | One byte | SBB C |
| DAD Rp | Adds the contents of the register pair to that of the H and L registers | Register direct | One byte | DAD B |
| INR R | Increments the register by 1 | Register direct | One byte | INR B |
| INX Rp | Increments the register pair by 1 | Register direct | One byte | INX B |
| DCR R | Decrements the register by 1 | Register direct | One byte | DCR E |
| DCX Rp | Decrements the register pair by 1 | Register direct | One byte | DCX D |
| ADD M | Adds the contents of the memory location pointed to by the HL register pair to that of the accumulator | Indirect | One byte | ADD M |
| ADC M | Adds the contents of the memory location pointed to by the HL register pair and the carry to that of the accumulator | Indirect | One byte | ADC M |
| SUB M | Subtracts the contents of the memory location pointed to by the HL register pair from that of the accumulator | Indirect | One byte | SUB M |
| SBB M | Subtracts the borrow and the contents of the memory location pointed to by the HL pair from that of the accumulator | Indirect | One byte | SBB M |

| | | | | |
|---|---|---|---|---|
| INR M | Increments the memory location pointed to by the HL register pair by 1 | Indirect | One byte | INR M |
| DCR M | Decrements the memory location pointed to by the HL register pair by 1 | Indirect | One byte | DCR M |
| DAA | Converts the contents of the accumulator from binary to BCD (Decimal-Adjust Accumulator) | Implicit | One byte | DAA |

The following points list some key features of arithmetic operations:

(i) For arithmetic operations, one of the data must be stored in the accumulator and the other given or addressed in the instruction.

(ii) Add-with-carry instructions are used for multi-byte and higher-order byte addition.

(iii) Similarly, subtract-with-borrow instructions are used in multi-byte and higher-order byte subtraction.

(iv) Increment and decrement instructions can be operated not only on the accumulator, but also on other registers including memory locations.

(v) The contents of a register pair can be incremented or decremented using INX and DCX instructions.

(vi) DAA is the 8085 instruction that supports BCD addition. The addition of BCD data is done like binary addition, using the ADD instruction. DAA is used to convert the result of the binary addition of BCD numbers into a BCD number. This instruction cannot be used to directly convert binary numbers into BCD numbers.

**Logical Instructions**

The most important logical instructions supported by the 8085 are AND, OR, EXOR, and NOT. The complete list is given in Table 1.10.

For logical operations, one of the data must be stored in the accumulator and the other given or addressed in the instruction. Logical operations can be performed with immediate data; data stored in a register, or indirectly addressed memory location content.

Besides the instructions already mentioned, two types of rotate instructions are available in the 8085. One set—RLC and RRC—rotates the accumulator contents within itself. The RLC instruction shifts the accumulator content left by one bit. In the process, the most significant bit of the accumulator becomes the least significant bit. The RRC instruction shifts the accumulator content right by one bit.

The other set of rotate instructions—RAL and RAR—rotates the accumulator content along with the carry flag. The RAL instruction shifts the accumulator content left by one bit and in the process, the most significant bit will be shifted to the carry flag and the carry flag content will be shifted to the least significant bit of the accumulator.

The instruction set of the 8085 supports a compare instruction for comparing the magnitude of two binary numbers. The compare instructions are used to compare the accumulator content with the operand specified in the instruction. CPI instruction uses immediate addressing and CMP uses registers or indirectly addressed memory location for comparing with the accumulator. The result of the compare instruction is indicated in the flag register, as follows:

If [(A) − operand] = 0, i.e., (A) = operand, the zero flag is set.

If [(A) − operand] < 0, i.e., (A) < operand, the carry flag is set.

If [(A) − operand] > 0, i.e., (A) > operand, the zero and carry flags are reset.

**Table 1.10** Logical instructions

| Mnemonics | Tasks performed on execution | Addressing mode | Instruction length | Example |
|---|---|---|---|---|
| ANI 8-bit | The 8-bit data is logically ANDed with the contents of the accumulator. | Immediate | Two bytes | ANI 0FH |
| XRI 8-bit | The 8-bit data is logically EXORed with the contents of the accumulator. | Immediate | Two bytes | XRI 01H |
| ORI 8-bit | The 8-bit data is logically ORed with the contents of the accumulator. | Immediate | Two bytes | ORI 80H |
| ANA R | The contents of the register are logically ANDed with the contents of the accumulator. | Register direct | One byte | ANA C |
| XRA R | The contents of the register are logically EXORed with the contents of the accumulator. | Register direct | One byte | XRA D |
| ORA R | The contents of the register are logically ORed with the contents of the accumulator. | Register direct | One byte | ORA E |
| ANA M | The contents of the memory location pointed to by the HL register pair is logically ANDed with the contents of the accumulator. | Indirect | One byte | ANA M |
| XRA M | The contents of the memory location pointed to by the HL register pair is logically EXORed with the contents of the accumulator. | Indirect | One byte | XRA M |
| ORA M | The contents of the memory location pointed to by the HL register pair is logically ORed with the contents of the accumulator. | Indirect | One byte | ORA M |
| RLC | Rotates the bits of the accumulator left by one position | Implicit | One byte | RLC |
| RRC | Rotates the bits of the accumulator right by one position | Implicit | One byte | RRC |
| RAL | Rotates the bits of the accumulator left by one position, through the carry | Implicit | One byte | RAL |
| RAR | Rotates the bits of the accumulator right by one position, through the carry | Implicit | One byte | RAR |
| CPI 8-bit | Compares the 8-bit data with the contents of the accumulator | Immediate | Two bytes | CPI FFH |
| CMP R | Compares the contents of the register with that of the accumulator | Register direct | One byte | CMP B |
| CMP M | Compares the contents of the memory location pointed to by the HL register pair with that of the accumulator | Indirect | One byte | CMP M |
| CMA | Complements the contents of the accumulator | Implicit | One byte | CMA |
| CMC | Complements the carry | Implicit | One byte | CMC |
| STC | Sets the carry | Implicit | One byte | STC |

**Branching Instructions**

Branching instructions are used to transfer the program execution to a different address. Branching instructions are of two types—jump instructions and subroutine instructions. The jump instructions merely transfer the execution from one location in the program to another, whereas the subroutine instructions in the main program transfer execution to a new location and also return to the main program. Return instructions are used for this purpose. The branching can take place unconditionally or conditionally, based on the flag conditions shown in Table 1.11. PCHL instruction is a special instruction used to branch to the address stored in the HL register pair.

RST *n* is the restart instruction supported by the 8085. Upon execution of the RST *n* instruction, the program execution will be transferred to the address given by $n \times 8$. For example, RST 4 instruction will transfer the execution to the address 0020H which is the hexadecimal equivalent of 32 (in decimal form).

In machine code or opcode, the16-bit or 4 hex digit addresses in the branching instructions are given such that the lower-order byte of the address follows the higher-order byte. For example, JMP 8030H is coded as C3 30 80. The opcode for JMP, C3, is stored first, followed by 30 and then by 80.

**Table 1.11** Branching instructions

| Mnemonics | Tasks performed on execution | Instruction length | Example |
|---|---|---|---|
| JMP 16-bit | Jump unconditionally | Three bytes | JMP 9500 |
| JC 16-bit | Jump if carry is set | Three bytes | JC 9500 |
| JNC 16-bit | Jump on no carry | Three bytes | JNC 9500 |
| JP 16-bit | Jump on positive | Three bytes | JP 9500 |
| JM 16-bit | Jump on minus | Three bytes | JM 9500 |
| JZ 16-bit | Jump on zero | Three bytes | JZ 9500 |
| JNZ16-bit | Jump on no zero | Three bytes | JNZ 9500 |
| JPE 16-bit | Jump on parity even | Three bytes | JPE 9500 |
| JPO 16-bit | Jump on parity odd | Three bytes | JPO 9500 |
| CALL 16-bit | Call unconditionally | Three bytes | CALL 9500 |
| CC 16-bit | Call on carry | Three bytes | CC 9500 |
| CNC 16-bit | Call on no carry | Three bytes | CNC 9500 |
| CP 16-bit | Call on positive | Three bytes | CP 9500 |
| CM 16-bit | Call on minus | Three bytes | CM 9500 |
| CZ 16-bit | Call on zero | Three bytes | CZ 9500 |
| CNZ 16-bit | Call on no zero | Three bytes | CNZ 9500 |
| CPE 16-bit | Call on parity even | Three bytes | CPE 9500 |
| CPO 16-bit | Call on parity odd | Three bytes | CPO 9500 |
| RET | Return unconditionally | One byte | RET |
| RC | Return on carry | One byte | RC |
| RNC | Return on no carry | One byte | RNC |
| RP | Return on positive | One byte | RP |
| RM | Return on minus | One byte | RM |
| RZ | Return on zero | One byte | RZ |
| RNZ | Return on no zero | One byte | RNZ |
| RPE | Return on parity even | One byte | RPE |
| RPO | Return on parity odd | One byte | RPO |
| PCHL | Copy HL contents to the program counter | One byte | PCHL |
| RST 0/1/2/3/4/5/6/7 | Restart | One byte | RST 5 |

**Machine Control Instructions**

Machine control instructions are used to control the microprocessor execution and functioning and are listed in Table 1.12. They are explained in detail in the following points:

i) NOP means no operation. When this instruction is executed, nothing is done; no changes occur in the contents of the registers. The program counter alone is incremented to fetch and execute the next instruction.

ii) HLT instruction is used to halt the execution of the program. The operation of the microprocessor is suspended when HLT instruction is executed. The only way to exit the halt state is to apply the hardware reset signal.

iii) Interrupts are disabled and enabled using DI and EI signals, respectively. Once the DI instruction has been executed, the processor ignores any interrupt request received. To enable interrupts again, the EI instruction has to be executed.

iv) The SIM instruction is used to send serial data on the serial output data (SOD) line of the microprocessor and the RIM instruction is used to receive serial data on the serial input data (SID) line of the processor. The SIM and RIM instructions are also associated with the setting and reading of interrupt masks for RST hardware interrupts.

**Table 1.12** Machine control instructions

| Mnemonics | Tasks performed on execution | Addressing mode | Instruction length |
|---|---|---|---|
| NOP | No operation | Implicit | One byte |
| HLT | Halts the microprocessor execution | Implicit | One byte |
| DI | Disables interrupts | Implicit | One byte |
| EI | Enables interrupts | Implicit | One byte |
| RIM | Reads interrupt mask | Implicit | One byte |
| SIM | Sets interrupt mask | Implicit | One byte |

**1.3 Introduction to 8051 Microcontrollers –**

The 8051 Microcontroller is one of the most popular and most commonly used microcontrollers in various fields like embedded systems, consumer electronics, automobiles, etc.

8051 Microcontroller has many features like Serial Communication, Timers, Interrupts, etc. and hence many students and beginners start their work on the concept of Microcontrollers with 8051 Microcontroller

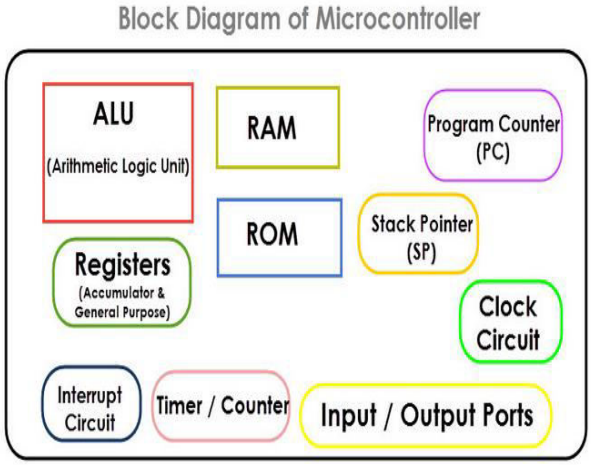
Even though 8051 Microcontroller might seem a little bit out of fashion, we feel that it is one of the best platforms to get started with Microcontrollers, Embedded Systems and Programming (both C and Assembly).

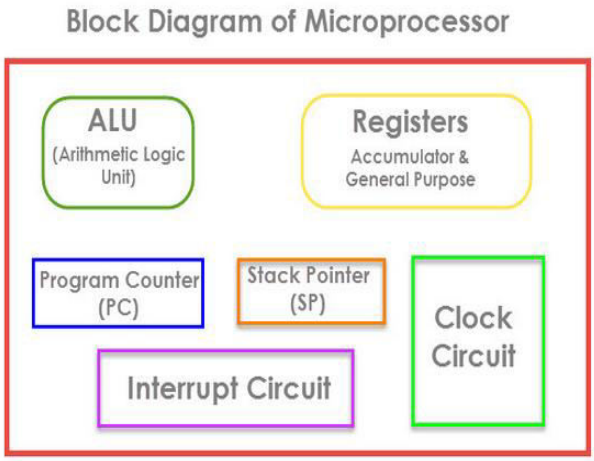
A Microcontroller is a VLSI IC that contains a CPU (Processor) along with some other peripherals like Memory (RAM and ROM), I/O Ports, Timers/Counters, Communication Interface, ADC, etc.

On the contrary, a Microprocessor (which was developed before Microcontroller) is just a Processor (CPU) and doesn't have the above mentioned peripherals. In order to make it work or build a system around it, we need to interface the peripherals separately.

Until the development of Microcontrollers, almost all process and control tasks were implemented using Microprocessors. As Microprocessor needs the additional peripherals to work as a system, the overall cost of the control system was high.

But with the development of Microcontroller, the situation has changed completely including the world of Embedded Systems.




Intel's 8051 Microcontroller (Intel MSC-51 Architecture) was a successor to 8048 Microcontroller (Intel MSC-48 Architecture). Originally, 8051 Microcontrollers were developed using N-MOS Technology but the use of battery powered devices and their low power consumption lead to usage of CMOS Technology (which is famous for its low power consumption).

**Applications of 8051 Microcontroller**

Even with the development of many advanced and superior Microcontrollers, 8051 Microcontroller is still being used in many embedded system and applications.

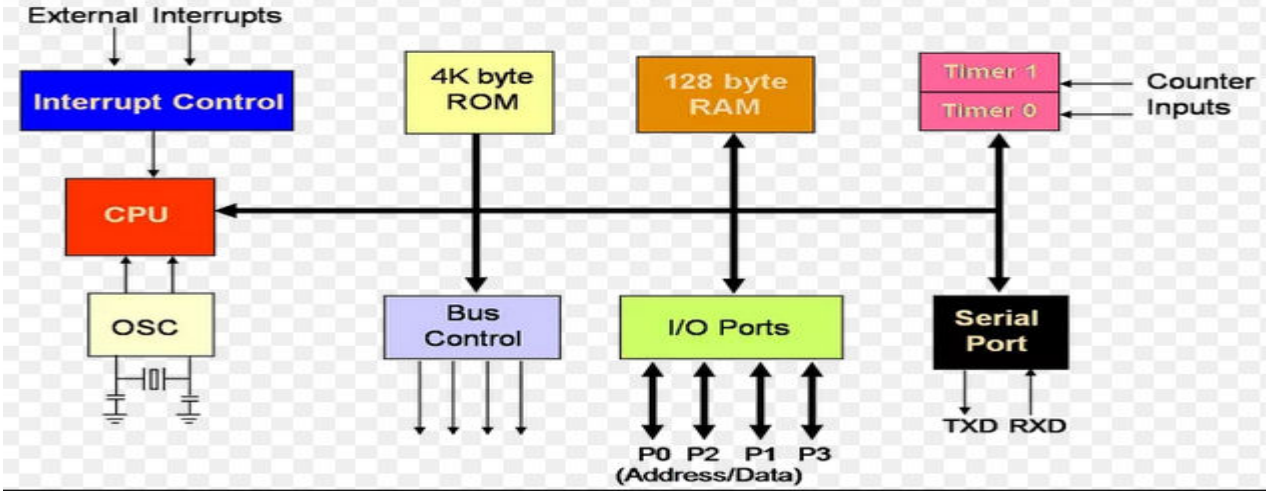
Some of the applications of 8051 Microcontroller are mentioned below:

- Consumer Appliances (TV Tuners, Remote controls, Computers, Sewing Machines, etc.)
- Home Applications (TVs, VCR, Video Games, Camcorder, Music Instruments, Home Security Systems, Garage Door Openers, etc.)
- Communication Systems (Mobile Phones, Intercoms, Answering Machines, Paging Devices, etc.)
- Office (Fax Machines, Printers, Copiers, Laser Printers, etc.)
- Automobiles (Air Bags, ABS, Engine Control, Transmission Control, Temperature Control, Keyless Entry)
- Aeronautical and Space
- Medical Equipment
- Defence Systems
- Robotics
- Industrial Process and Flow Control
- Radio and Networking Equipment
- Remote Sensing

**8051 Microcontroller Basics**

8051 is an 8 – bit Microcontroller i.e. the data bus of the 8051 Microcontroller (both internal and external) is 8 – bit wide. It is a CISC based Microcontroller with Harvard Architecture (separate program and data memory).

Since the basic layout of a microcontroller includes a CPU, ROM, RAM, etc. the 8051 microcontroller also has a similar layout.




***8051 Microcontroller Features***

- ***8–Bit ALU***: ALU or Arithmetic Logic Unit is the heart of a microcontroller. It performs arithmetic and bitwise operation on binary numbers. The ALU in 8051 is an 8 – Bit ALU i.e. it can perform operations on 8 – bit data.
- ***8–Bit Accumulator***: The Accumulator is an important register associated with the ALU. The accumulator in 8051 is an 8 – bit register.
- ***RAM***: 8051 Microcontroller has 128 Bytes of RAM which includes SFRs and Input/Output Port Registers.
- ***ROM***: 8051 has 4 KB of on-chip ROM (Program Memory).
- ***I/O Ports***: 8051 has four 8 – bit Input / Output Ports which are bit addressable and bidirectional.
- ***Timers / Counters***: 8051 has two 16 – bit Timers / Counters.
- ***Serial Port***: 8051 supports full duplex UART Communication.
- ***External Memory***: 8051Microcontroller can access two 16 – bit address line at once: one each for RAM and ROM. The total external memory that an 8051 Microcontroller can access for RAM and ROM is 64KB ($2^{16}$ for each type).
- ***Additional Features***: Interrupts, on-chip oscillator, Boolean Processor, Power Down Mode, etc.
- An oscillator or even a crystal clock is required in a **microcontroller**, when it is synchronizing the controls of the independent circuit it is connected to, from component layer to machine layer

**8051 Microcontroller Memory Organization**

The 8051 Microcontroller Memory is separated in Program Memory (ROM) and Data Memory (RAM). The Program Memory of the 8051 Microcontroller is used for storing the program to be executed i.e. instructions. The Data Memory on the other hand, is used for storing temporary variable data and intermediate results.

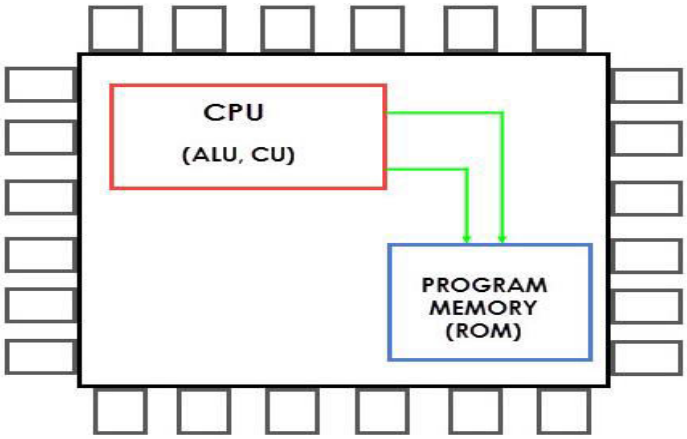
8051 Microcontroller has both Internal ROM and Internal RAM. If the internal memory is inadequate, you can add external memory using suitable circuits.

***Program Memory (ROM) of 8051 Microcontroller***

In 8051 Microcontroller, the code or instructions to be executed are stored in the Program Memory, which is also called as the ROM of the Microcontroller. The original 8051 Microcontroller by Intel has 4KB of internal ROM.

Some variants of 8051 like the 8031 and 8032 series doesn’t have any internal ROM (Program Memory) and must be interfaced with external Program Memory with instructions loaded in it.

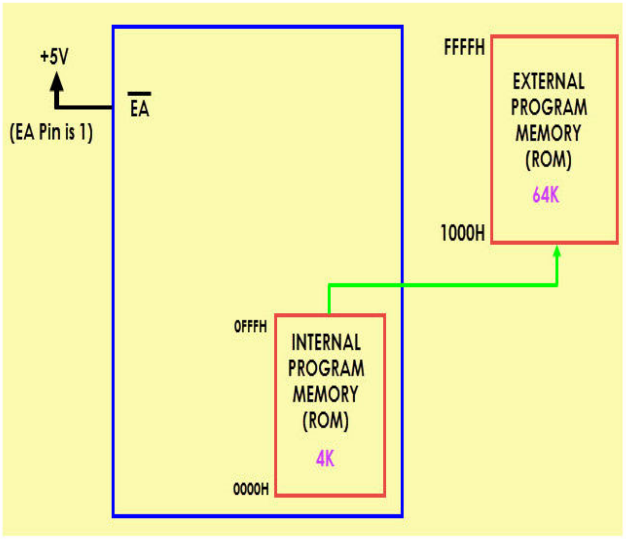
Almost all modern 8051 Microcontrollers, like 8052 Series, have 8KB of Internal Program Memory (ROM) in the form of Flash Memory (ROM) and provide the option of reprogramming the memory.

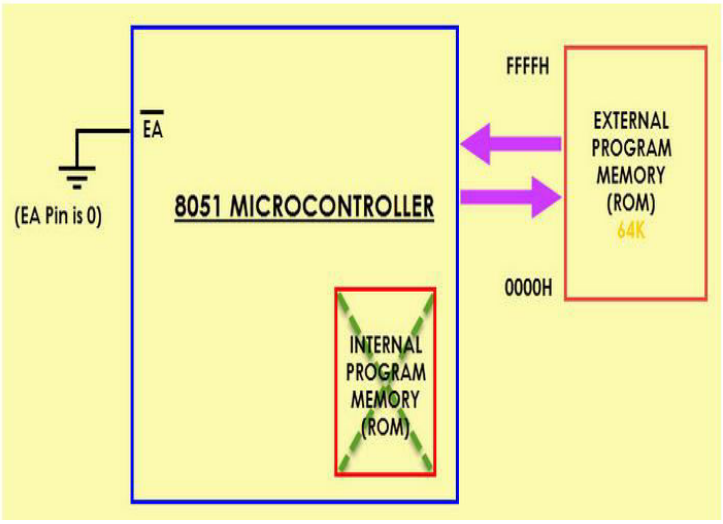



In case of 4KB of Internal ROM, the address space is 0000H to 0FFFH. If the address space i.e. the program addresses exceed this value, then the CPU will automatically fetch the code from the external Program Memory.

For this, the External Access Pin (EA Pin) must be pulled HIGH i.e. when the EA Pin is high, the CPU first fetches instructions from the Internal Program Memory in the address range of 0000H to 0FFFFH and if the memory addresses exceed the limit, then the instructions are fetched from the external ROM in the address range of 1000H to FFFFH.

There is another way to fetch the instructions: ignore the Internal ROM and fetch all the instructions only from the External Program Memory (External ROM). For this scenario, the EA Pin must be connected to GND. In this case, the memory addresses of the external ROM will be from 0000H to FFFFH.




***Data Memory (RAM) of 8051 Microcontroller***

The Data Memory or RAM of the 8051 Microcontroller stores temporary data and intermediate results that are generated and used during the normal operation of the microcontroller. Original Intel's 8051 Microcontroller had 128B of internal RAM.

But almost all modern variants of 8051 Microcontroller have 256B of RAM. In this 256B, the first 128B i.e. memory addresses from 00H to 7FH is divided in to Working Registers (organized as Register Banks), Bit – Addressable Area and General Purpose RAM (also known as Scratchpad area).

In the first 128B of RAM (from 00H to 7FH), the first 32B i.e. memory from addresses 00H to 1FH consists of 32 Working Registers that are organized as four banks with 8 Registers in each Bank.

The 4 banks are named as Bank0, Bank1, Bank2 and Bank3. Each Bank consists of 8 registers named as R0 – R7. Each Register can be addressed in two ways: either by name or by address.

To address the register by name, first the corresponding Bank must be selected. In order to select the bank, we have to use the RS0 and RS1 bits of the Program Status Word (PSW) Register (RS0 and RS1 are 3rd and 4th bits in the PSW Register).

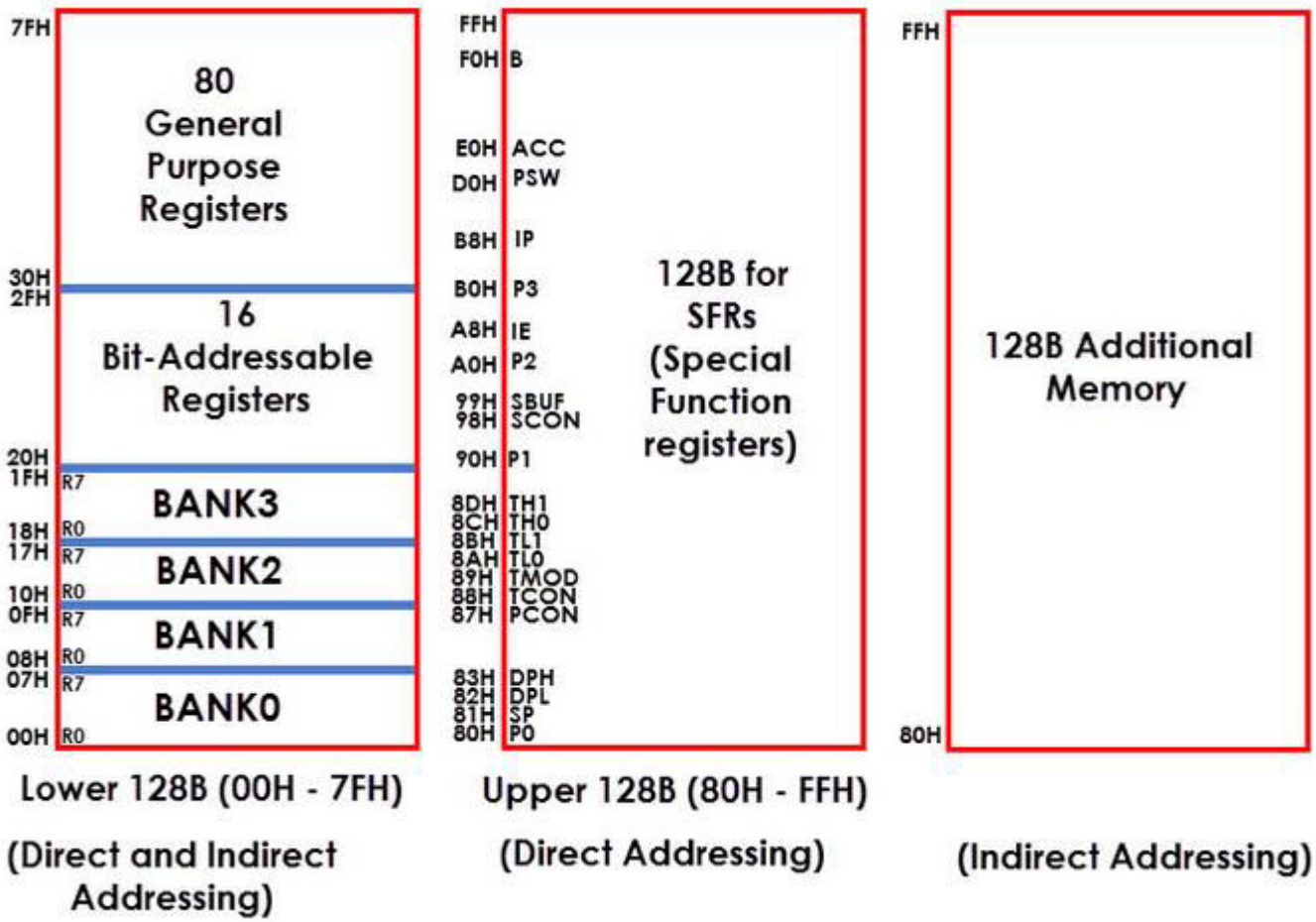
When addressing the Register using its address i.e. 12H for example, the corresponding Bank may or may not be selected. (12H corresponds to R2 in Bank2).

The next 16B of the RAM i.e. from 20H to 2FH are Bit – Addressable memory locations. There are totally 128 bits that can be addressed individually using 00H to 7FH or the entire byte can be addressed as 20H to 2FH. For example 32H is the bit 2 of the internal RAM location 26H.

The final 80B of the internal RAM i.e. addresses from 30H to 7FH, is the general purpose RAM area which are byte addressable.

These lower 128B of RAM can be addressed directly or indirectly.

The upper 128B of the RAM i.e. memory addresses from 80H to FFH is allocated for Special Function Registers (SFRs). SFRs control specific functions of the 8051 Microcontroller. Some of the SFRs are I/O Port Registers (P0, P1, P2 and P3), PSW (Program Status Word), A (Accumulator), IE (Interrupt Enable), PCON (Power Control), etc.



| *Name of the Register* | *Function* | *Internal RAM Address (HEX)* |
|---|---|---|
| ACC | Accumulator | E0H |
| B | B Register (for Arithmetic) | F0H |
| DPH | Addressing External Memory | 83H |
| DPL | Addressing External Memory | 82H |
| IE | Interrupt Enable Control | A8H |
| IP | Interrupt Priority | B8H |
| P0 | PORT 0 Latch | 80H |
| P1 | PORT 1 Latch | 90H |
| P2 | PORT 2 Latch | A0H |
| P3 | PORT 3 Latch | B0H |
| PCON | Power Control | 87H |
| PSW | Program Status Word | D0H |
| SCON | Serial Port Control | 98H |
| SBUF | Serial Port Data Buffer | 99H |
| SP | Stack Pointer | 81H |
| TMOD | Timer / Counter Mode Control | 89H |
| TCON | Timer / Counter Control | 88H |
| TL0 | Timer 0 LOW Byte | 8AH |
| TH0 | Timer 0 HIGH Byte | 8CH |
| TL1 | Timer 1 LOW Byte | 8BH |
| TH1 | Timer 1 HIGH Byte | 8DH |