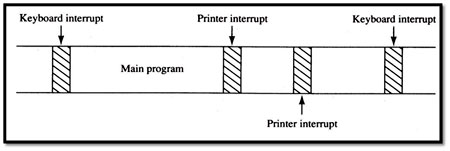
**Need of Interrupts ?**

As we studied, the Microprocessor can serve several devices. There are two ways to offer service: Interrupts and Polling.

* The advantage of interrupts is that the microprocessor can serve many devices (not all at the same time, of course); each device can get the attention of the microprocessor based on the priority assigned to it.
* The polling method cannot assign priority because it checks all devices in a round-robin fashion.
* More importantly, in the interrupt method the microprocessor can also ignore (mask) a device request for service.
* This is not possible with the polling method.
* The most important reason that the interrupt method is preferable is that the polling method wastes much of the microprocessor’s time by polling devices that do not need service.
* So interrupts are used to avoid tying down the microprocessor.

To understand the difference better, consider this example. The polling method is very much similar to a salesperson. The salesman goes door-to-door requesting to buy his product. Like processor keeps monitoring the flags or signals one by one for all devices. Interrupt is very similar to a shopkeeper. Whosever needs a service or product goes to him and approaches him. Like, when the flags or signals are received, they notify the processor that they need its service.  
  
**Interrupts are useful when interfacing I/O devices with low data-transfer rates, like a keyboard or a mouse, in which case polling the device wastes valuable processing time**  
  
Above time line shows typing on a keyboard, a printer removing data from memory, and a program executing. The keyboard interrupt service procedure, called by the keyboard interrupt, and the printer interrupt service procedure each take little time to execute

Unit -3

Q) Microprocessor - 8086 Instruction Sets

The 8086 microprocessor supports 8 types of instructions −

* Data Transfer Instructions
* Arithmetic Instructions
* Bit Manipulation Instructions
* String Instructions
* Program Execution Transfer Instructions (Branch & Loop Instructions)
* Processor Control Instructions
* Iteration Control Instructions
* Interrupt Instructions

Let us now discuss these instruction sets in detail.

**Data Transfer Instructions**

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

**Instruction to transfer a word**

* **MOV** − Used to copy the byte or word from the provided source to the provided destination.
* **PPUSH** − Used to put a word at the top of the stack.
* **POP** − Used to get a word from the top of the stack to the provided location.
* **PUSHA** − Used to put all the registers into the stack.
* **POPA** − Used to get words from the stack to all registers.
* **XCHG** − Used to exchange the data from two locations.
* **XLAT** − Used to translate a byte in AL using a table in the memory.

**Instructions for input and output port transfer**

* **IN** − Used to read a byte or word from the provided port to the accumulator.
* **OUT** − Used to send out a byte or word from the accumulator to the provided port.

**Instructions to transfer the address**

* **LEA** − Used to load the address of operand into the provided register.
* **LDS** − Used to load DS register and other provided register from the memory
* **LES** − Used to load ES register and other provided register from the memory.

**Instructions to transfer flag registers**

* **LAHF** − Used to load AH with the low byte of the flag register.
* **SAHF** − Used to store AH register to low byte of the flag register.
* **PUSHF** − Used to copy the flag register at the top of the stack.
* **POPF** − Used to copy a word at the top of the stack to the flag register.

**Arithmetic Instructions**

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.

Following is the list of instructions under this group −

**Instructions to perform addition**

* **ADD** − Used to add the provided byte to byte/word to word.
* **ADC** − Used to add with carry.
* **INC** − Used to increment the provided byte/word by 1.
* **AAA** − Used to adjust ASCII after addition.
* **DAA** − Used to adjust the decimal after the addition/subtraction operation.

**Instructions to perform subtraction**

* **SUB** − Used to subtract the byte from byte/word from word.
* **SBB** − Used to perform subtraction with borrow.
* **DEC** − Used to decrement the provided byte/word by 1.
* **NPG** − Used to negate each bit of the provided byte/word and add 1/2’s complement.
* **CMP** − Used to compare 2 provided byte/word.
* **AAS** − Used to adjust ASCII codes after subtraction.
* **DAS** − Used to adjust decimal after subtraction.

**Instruction to perform multiplication**

* **MUL** − Used to multiply unsigned byte by byte/word by word.
* **IMUL** − Used to multiply signed byte by byte/word by word.
* **AAM** − Used to adjust ASCII codes after multiplication.

**Instructions to perform division**

* **DIV** − Used to divide the unsigned word by byte or unsigned double word by word.
* **IDIV** − Used to divide the signed word by byte or signed double word by word.
* **AAD** − Used to adjust ASCII codes after division.
* **CBW** − Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
* **CWD** − Used to fill the upper word of the double word with the sign bit of the lower word.

**Bit Manipulation Instructions**

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.

Following is the list of instructions under this group −

**Instructions to perform logical operation**

* **NOT** − Used to invert each bit of a byte or word.
* **AND** − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
* **OR** − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
* **XOR** − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
* **TEST** − Used to add operands to update flags, without affecting operands.

**Instructions to perform shift operations**

* **SHL/SAL** − Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
* **SHR** − Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
* **SAR** − Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

**Instructions to perform rotate operations**

* **ROL** − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
* **ROR** − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
* **RCR** − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
* **RCL** − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

**String Instructions**

String is a group of bytes/words and their memory is always allocated in a sequential order.

Following is the list of instructions under this group −

* **REP** − Used to repeat the given instruction till CX ≠ 0.
* **REPE/REPZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
* **REPNE/REPNZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
* **MOVS/MOVSB/MOVSW** − Used to move the byte/word from one string to another.
* **COMS/COMPSB/COMPSW** − Used to compare two string bytes/words.
* **INS/INSB/INSW** − Used as an input string/byte/word from the I/O port to the provided memory location.
* **OUTS/OUTSB/OUTSW** − Used as an output string/byte/word from the provided memory location to the I/O port.
* **SCAS/SCASB/SCASW** − Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
* **LODS/LODSB/LODSW** − Used to store the string byte into AL or string word into AX.

**Program Execution Transfer Instructions (Branch and Loop Instructions)**

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

Instructions to transfer the instruction during an execution without any condition −

* **CALL** − Used to call a procedure and save their return address to the stack.
* **RET** − Used to return from the procedure to the main program.
* **JMP** − Used to jump to the provided address to proceed to the next instruction.
* Instructions to transfer the instruction during an execution with some conditions −
* **JA/JNBE** − Used to jump if above/not below/equal instruction satisfies.
* **JAE/JNB** − Used to jump if above/not below instruction satisfies.
* **JBE/JNA** − Used to jump if below/equal/ not above instruction satisfies.
* **JC** − Used to jump if carry flag CF = 1
* **JE/JZ** − Used to jump if equal/zero flag ZF = 1
* **JG/JNLE** − Used to jump if greater/not less than/equal instruction satisfies.
* **JGE/JNL** − Used to jump if greater than/equal/not less than instruction satisfies.
* **JL/JNGE** − Used to jump if less than/not greater than/equal instruction satisfies.
* **JLE/JNG** − Used to jump if less than/equal/if not greater than instruction satisfies.
* **JNC** − Used to jump if no carry flag (CF = 0)
* **JNE/JNZ** − Used to jump if not equal/zero flag ZF = 0
* **JNO** − Used to jump if no overflow flag OF = 0
* **JNP/JPO** − Used to jump if not parity/parity odd PF = 0
* **JNS** − Used to jump if not sign SF = 0
* **JO** − Used to jump if overflow flag OF = 1
* **JP/JPE** − Used to jump if parity/parity even PF = 1
* **JS** − Used to jump if sign flag SF = 1

**Processor Control Instructions**

These instructions are used to control the processor action by setting/resetting the flag values.

Following are the instructions under this group −

* **STC** − Used to set carry flag CF to 1
* **CLC** − Used to clear/reset carry flag CF to 0
* **CMC** − Used to put complement at the state of carry flag CF.
* **STD** − Used to set the direction flag DF to 1
* **CLD** − Used to clear/reset the direction flag DF to 0
* **STI** − Used to set the interrupt enable flag to 1, i.e., enable INTR input.
* **CLI** − Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

**Iteration Control Instructions**

These instructions are used to execute the given instructions for number of times. Following is the list of instructions under this group −

* **LOOP** − Used to loop a group of instructions until the condition satisfies, i.e., CX = 0
* **LOOPE/LOOPZ** − Used to loop a group of instructions till it satisfies ZF = 1 & CX = 0
* **LOOPNE/LOOPNZ** − Used to loop a group of instructions till it satisfies ZF = 0 & CX = 0
* **JCXZ** − Used to jump to the provided address if CX = 0

**Interrupt Instructions**

These instructions are used to call the interrupt during program execution.

* **INT** − Used to interrupt the program during execution and calling service specified.
* **INTO** − Used to interrupt the program during execution if OF = 1
* **IRET** − Used to return from interrupt service to the main program

Q) Microprocessor - 8086 Addressing Modes ?

The different ways in which a source operand is denoted in an instruction is known as **addressing modes**. There are 8 different addressing modes in 8086 programming −

## **Immediate addressing mode**

The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.

### **Example**

MOV CX, 4929 H, ADD AX, 2387 H, MOV AL, FFH

## **Register addressing mode**

It means that the register is the source of an operand for an instruction.

### **Example**

MOV CX, AX ; copies the contents of the 16-bit AX register into

; the 16-bit CX register),

ADD BX, AX

## **Direct addressing mode**

The addressing mode in which the effective address of the memory location is written directly in the instruction.

### **Example**

MOV AX, [1592H], MOV AL, [0300H]

## **Register indirect addressing mode**

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.

### **Example**

MOV AX, [BX] ; Suppose the register BX contains 4895H, then the contents

; 4895H are moved to AX

ADD CX, {BX}

## **Based addressing mode**

In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.

### **Example**

MOV DX, [BX+04], ADD CL, [BX+08]

## **Indexed addressing mode**

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.

### **Example**

MOV BX, [SI+16], ADD AL, [DI+16]

## **Based-index addressing mode**

In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register.

### **Example**

ADD CX, [AX+SI], MOV AX, [AX+DI]

## **Based indexed with displacement mode**

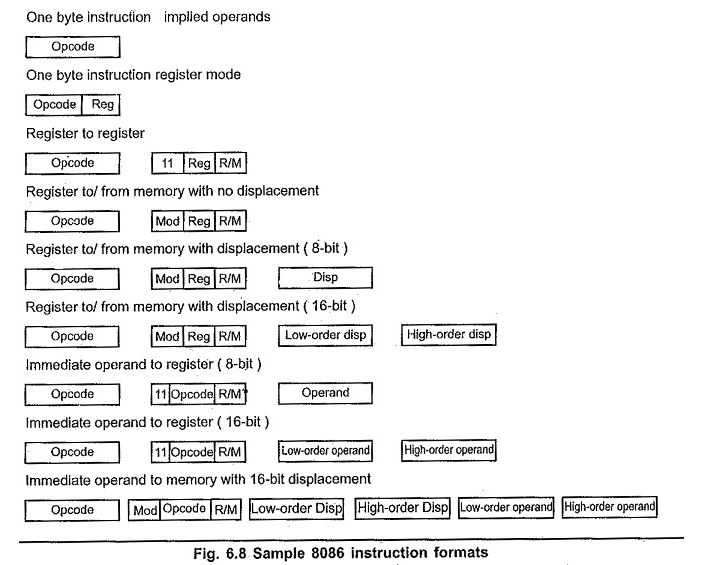
In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

### **Example**

MOV AX, [BX+DI+08], ADD CX, [BX+SI+16]

## Q) **8086 Instruction Format ?**

The 8086 Instruction 8086 Instruction Format vary from 1 to 6 bytes in length. Fig. 6.8 shows the instruction formats for 1 to 6 bytes instructions. As shown in the Fig. 6.8, displacements and operands may be either 8-bits or 16-bits long depending on the instruction. The opcode and the addressing mode is specified using first two bytes of an instruction



The opcode/addressing mode byte(s) may be followed by :

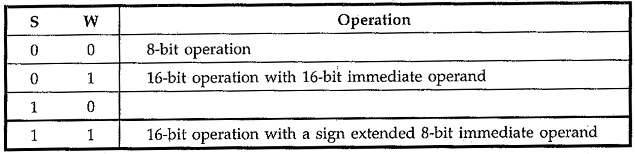
* **No additional byte**
* **Two byte EA (For direct addressing only).**
* **One or two byte displacement**
* **One or two byte immediate operand**
* **One or two byte displacement followed by a one or two byte immediate operand**
* **Two byte displacement and a two byte segment address (for direct intersegment addressing only).**

Most of the opcodes in 8086 has a special 1-bit indicates. They are :

**W-bit :** Some instructions of 8086 can operate on byte or a word. The W-bit in the opcode of such instruction specify whether instruction is a byte instruction (W = 0) or a word instruction (W = 1).

**D-bit :** The D-bit in the opcode of the instruction indicates that the register specified within the instruction is a source register (D = 0) or destination register (D =1).

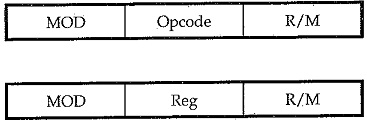
**S-bit :** An 8-bit 2’s complement number can be extended to a 16-bit 2’s complement number by making all of the bits in the higher-order byte equal the most significant bit in the low order byte. This is known as sign extension. The S-bit along with the W-bit indicate :



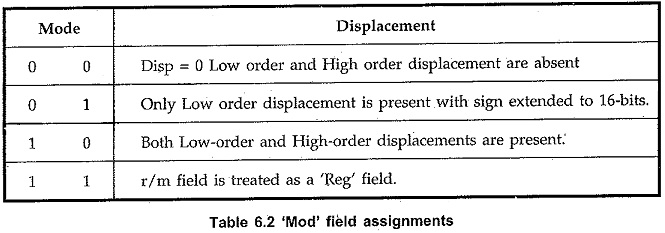
**V-bit :** V-bit decides the number of shifts for rotate and shift instructions. If V = 0, then count = 1; if V = 1, the count is in CL register. For example, if V = 1 and CL = 2 then [shift](http://www.allaboutcircuits.com/) or rotate instruction shifts or rotates 2-bits

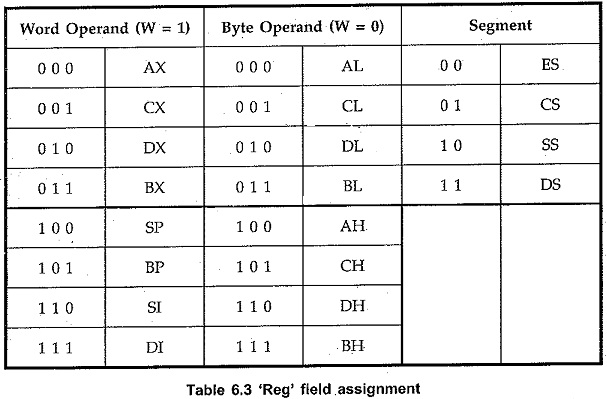
**Z-bit :** It is used for string primitives such as REP for comparison with ZF Flag. (Refer Appendix A for instruction formats)

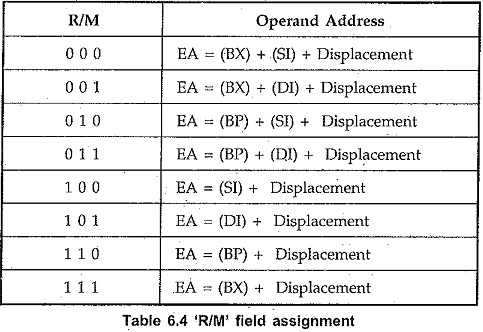
As seen from the Fig. 6.8 if an instruction has two opcode/addressing mode bytes, then the second byte is of one of the following two forms .

[](https://www.eeeguide.com/wp-content/uploads/2018/08/8086-Instruction-Set-2.jpg)

where Mod, Reg and R/M fields specify operand as described in the following tables.

[](https://www.eeeguide.com/wp-content/uploads/2018/08/8086-Instruction-Set-3.jpg)

[](https://www.eeeguide.com/wp-content/uploads/2018/08/8086-Instruction-Set-4.jpg)

[](https://www.eeeguide.com/wp-content/uploads/2018/08/8086-Instruction-Set-5.jpg)