SVKM'S NMIMS, School of Technology Management & Engineering | Navi-Mumbai

MBA-Tech (A.Y. 2020-21)

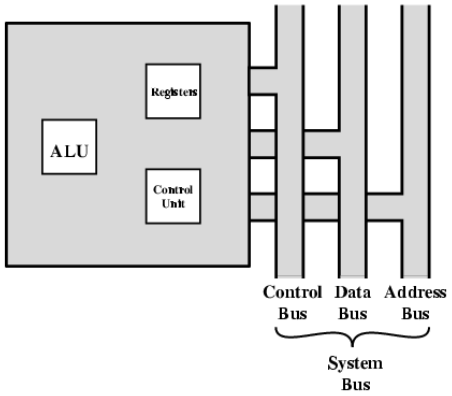
Lecture Notes

(Introduction to X86-Central Processing Unit)

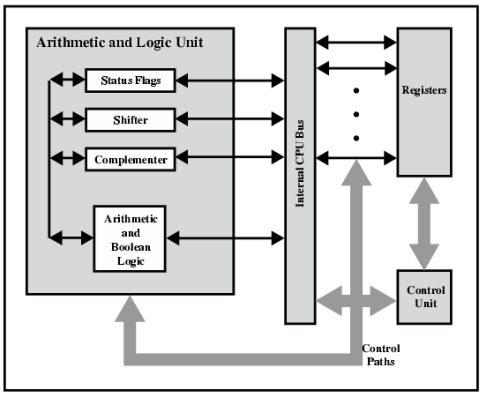
PROCESSOR ORGANISATION

- To understand the organization of the processor, let us consider the requirements placed on the processor, the things that it must do:
- Fetch instruction: The processor reads an instruction from memory (register, cache, main memory).
- Interpret instruction: The instruction is decoded to determine what action is required.
- Fetch data: The execution of an instruction may require reading data from memory or an I/O module.
- Process data: The execution of an instruction may require performing some arithmetic or logical operation on data.
- Write data: The results of an execution may require writing data to memory or an I/O module.

To do these things, it should be clear that the processor needs to store some data temporarily. In other words, the processor needs a small internal memory.
Figure.1 is a simplified view of a processor, indicating its connection to the rest of the system via the system bus.
The major components of the processor are an arithmetic and logic unit (ALU) and a control unit (CU). The ALU does the actual computation or processing of data.



The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU. In addition, the figure shows a minimal internal memory, consisting of a set of storage locations, called registers.
Figure.2 is a slightly more detailed view of the processor.
The data transfer and logic control paths are indicated, including internal processor bus which is needed to transfer data between the various registers and the ALU because the ALU in fact operates only on data in the internal processor memory.



- **Register Organization:**

In the processor register function as a level of memory above cache and main memory in the hierarchy. There are two main categories of registers.

(a) User Visible registers
(b) Control and Status registers

- (a) User Visible registers: These registers are used by the processor and programmer for minimizing the usage of memory references. These registers are further classified as:

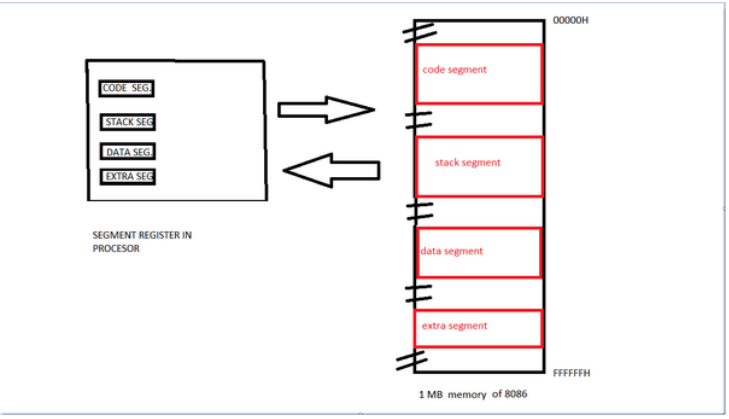
  - General Purpose registers
  - Data Registers and Address register
  - Conditional codes

(i) General Purpose Registers:

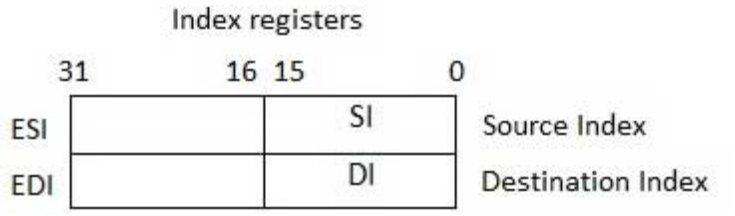
- These registers are used to hold operand for opcode, but there are certain restrictions like some dedicated registers are used for floating point and stack operations.
- These registers are also used for addressing functions.

(ii) Data and Address registers:

- Data registers are used to hold explicitly the data used for processing, while address registers are used as general purpose as well as address pointers in some addressing modes.
- **Address registers** includes: segment registers, Index registers, and Stack pointers.
- **Segment registers**: These registers hold the segment base address in segmented addressing.



- **Index registers:** These are used to hold index address in indexing addressing mode.



- **Stack pointer**: This is dedicated register used as pointer to point to the top of the stack memory.

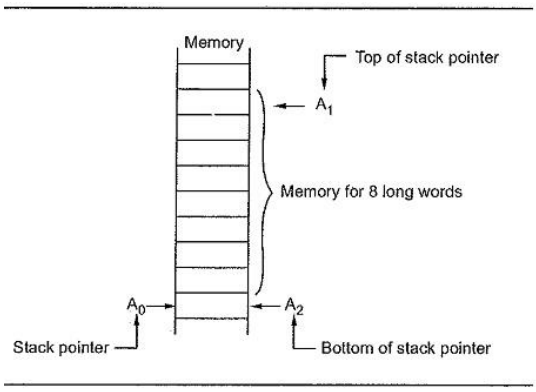


Fig. 11.11 Software controlled stack structure

(iii) Control Codes:

- These are partially visible to the programmer. These registers are used to hold the conditional bits which are set and reset by the processor hardware to indicate the status of the operation.'
- Implicit machine instructions are used to read the status of the control code. Some machine instructions use control word to test the condition as a part of branch operation.

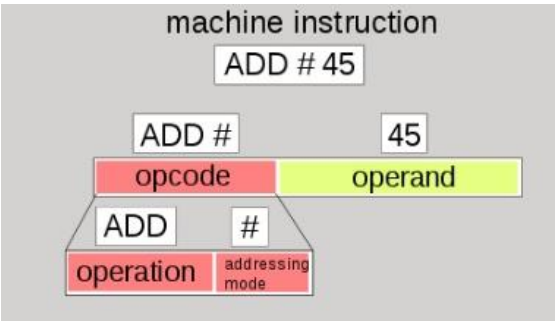
- (b) Control Register and Status: These registers are used to control the operation of the processor, most of these registers are invisible to the programmer and used by the processor. Some of the control registers are visible in machine control or operating system modes. Basic four registers are present in control and status group are:
  - Program counter (PC)
  - Instruction register (IR)
  - Memory address register (MAR)
  - Memory buffer register (MBR)

(i) Program register (PC): It is used to hold the address of the next instruction to be fetched from the memory.
(ii) Instruction register (IR): It holds the instruction most recently fetched from the memory.
(iii) Memory address registers (MAR): It holds the address of the memory location.
(iv) Memory buffer register (MBR): It is a buffer which holds the data to be written to memory or most recently read from memory.

Most of the processor contains set of registers which are used to indicate status information these are called as program status word (PSW). PSW contains conditional codes plus status information commonly used fields are sign, zero, carry, equal, overflow, interrupt enable/disable, supervisor.

Instruction Format:

- Every computing machine requires instructions, which define the operations to be performed on data.
- The instructions are represented in the form of binary bits in which every bit has its own significance, the layout of each bit in terms of its fields position is called as instruction format.
- It consists of an Opcode field which is a mandatory field and operand field which is flexible in size.
- The opcode field indicates the type of operation to be performed by the instruction where as operand field indicates the method in which the data is obtained and operated on.
- In short, **Opcode** is a part of the instruction that tells the processor what should be done. **Operand** is a part of the instruction that contains the data to be acted on, or the memory location of the data in a register.

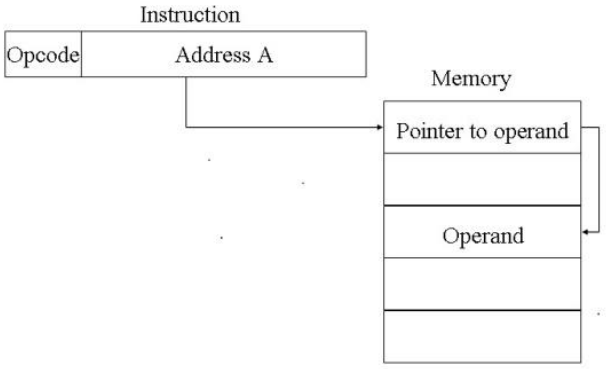


Addressing Modes: Describing the operand and its location is referred to as addressing modes. Following are the list of commonly used addressing modes:

(a) Immediate
(b) Direct
(c) Indirect
(d) Register
(e) Register indirect
(f) Displacement
(g) Stack

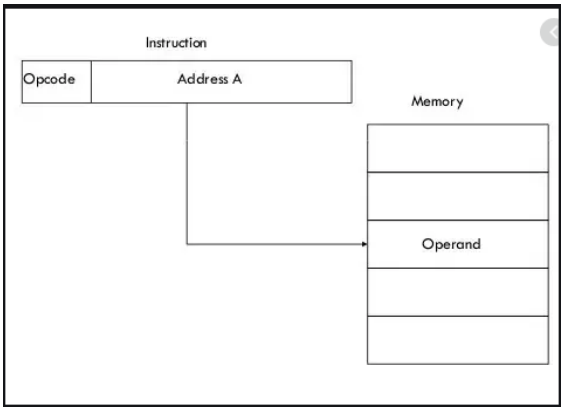
a) Immediate Addressing mode:

- In this addressing mode, the operand value is present as a part of the instruction.
- This mode is used for accessing constants and set values of variables.
- The advantage of this addressing mode it requires no memory reference for accessing the operand.
- Generally the operand value is stored in two's complement form.



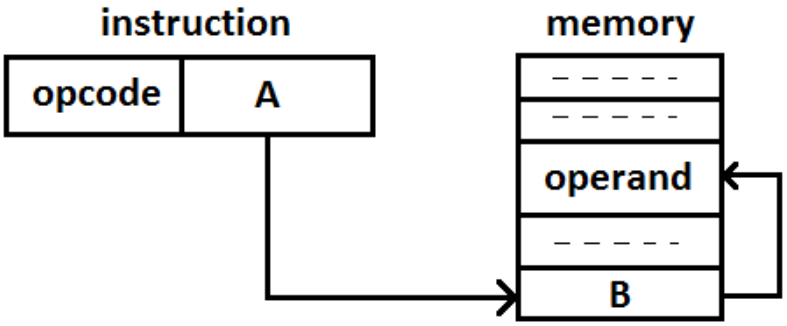
b) Direct Addressing mode:

- In this addressing mode the effective address of the operand is present as a part of the instruction. It is a simplest form of addressing mode as it requires only one memory reference.
- The major disadvantage of this addressing mode it can access limited range of address.



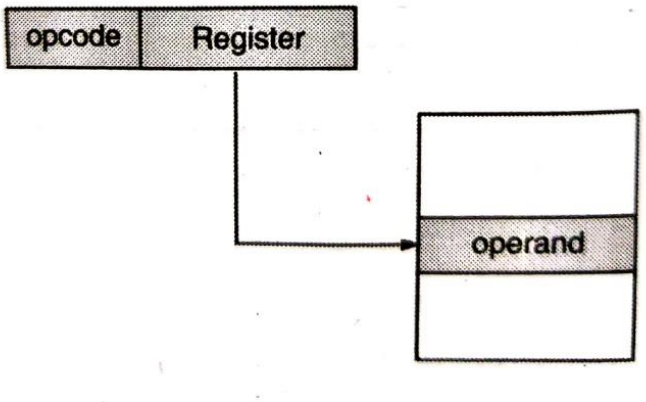
c) Indirect Addressing mode:

- In this addressing mode the operand address is obtained from word of memory addressed by the instruction address field.
- The main advantage of this addressing mode is that it can access large address range.
- Let say if N number of bits can be stored at one location of memory then the total available space is 2^N.
- The disadvantage of this mode is that it requires two memory references to fetch the operand.



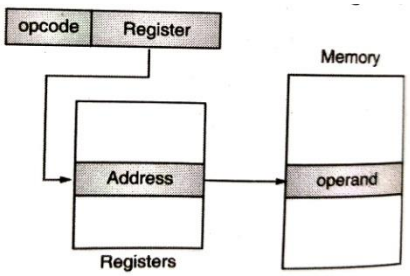
d) Register Addressing mode:

- In this addressing mode the operands are placed in the register and the address of register is indicated in the instruction.
- There are 3 to 5 bits for address field of register which in turn provides reference to 8 to 32 general purpose registers.
- The main advantage of this register is no memory references hence it's a fast transfer, small address field.
- The disadvantage of this mode is that it has limited number of registers which requires greater management efforts on programmer side.



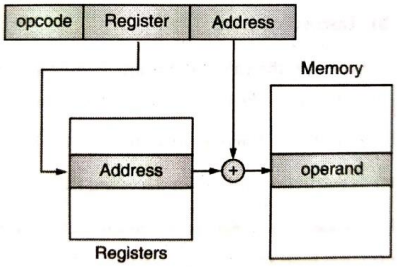
e) Register Indirect Addressing mode:

- The address of the operand is provided by the register instead of memory.
- The space available in this mode is proportional to the length of the register.
- The main advantage of this mode is it requires only one memory fetch cycle.



f) Displacement addressing mode:

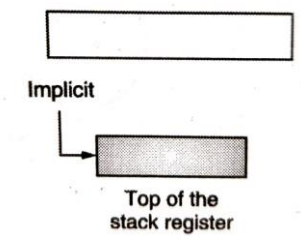
- This addressing mode requires two address fields in which one is explicit
- The other implicit reference address refers to a register whose content value is added to the explicit address to form the effective address of the operand.



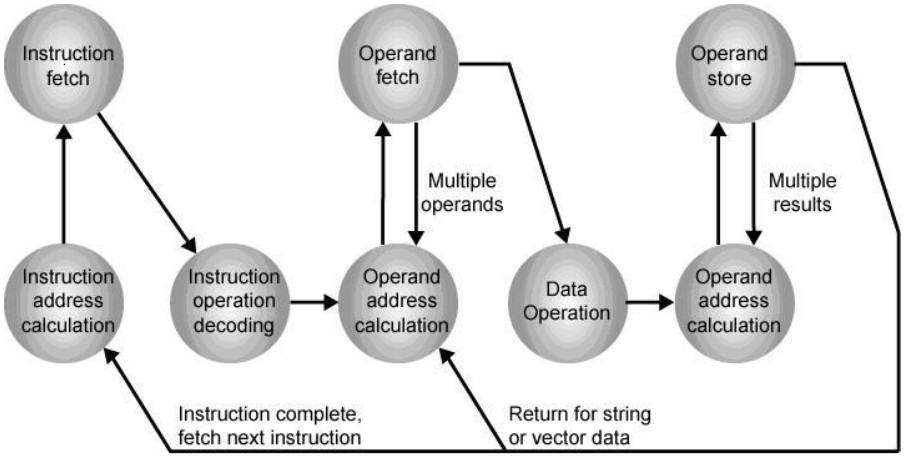
(i) Relative Addressing: In this addressing mode the implicit address is given by the program counter i.e. the address field is added with the next instruction address to form the effective address.
(ii) Base Register Addressing: In this addressing mode the effective address is formed by adding address field displacement value to the referenced register value. The register reference may be explicit or implicit.
(iii) Indexing: In this mode the effective address is formed by adding address field to a positive displacement value given by referenced register.

g) Stack Addressing:

- This addressing mode is an Implicit addressing mode in which no memory reference is mentioned. The address of the operand is always the top of the stack.



- **The Instruction cycle:**



The description for each stage of instruction cycle is as follows:

(a) **Instruction address calculation (IAC):** In this stage the address of the next instruction is computed by adding a fixed number to the previous instruction address.

(b) **Instruction fetch (IF):** In this stage, the calculated instruction address is read from the memory system and is sent to the processor or CPU.

(c) **Instruction operation decoding (ID):** In this stage the meaning of the instruction is extracted which determines the type of operation to be performed and operands to be used. This process is called as Instruction decoding. It is carried out in the part of control unit, conventionally called as Instruction Decoder.

(d) **Operand address calculation (OAC):** The address of the operand is computed that is to be fetched from memory or I/O space.

(e) **Operand fetch (OF):** In this stage, the operand is fetched from the memory or is read from an I/O device by sending the operand address computed in the previous stage, over the address bus and receiving operand over the data bus.

(f) **Data operation (DO or EXEC):** This stage performs operation indicated in the instruction. Therefore, this is the stage where the instruction currently under processing is actually executed.

(g) **Operand store (OS):** Writes the result into the memory or out to I/O.

8086 (X86) Processor:

Processor operations mostly involve processing data. This data can be stored in memory and accessed from thereon. However, reading data from and storing data into memory slows down the processor, as it involves complicated processes of sending the data request across the control bus and into the memory storage unit and getting the data through the same channel.

To speed up the processor operations, the processor includes some internal memory storage locations, called **registers**.

The registers store data elements for processing without having to access the memory. A limited number of registers are built into the processor chip.

Processor Registers:

There are ten 32-bit and six 16-bit processor registers in IA-32 architecture. The registers are grouped into three categories −

- General registers,
- Control registers, and
- Segment registers.

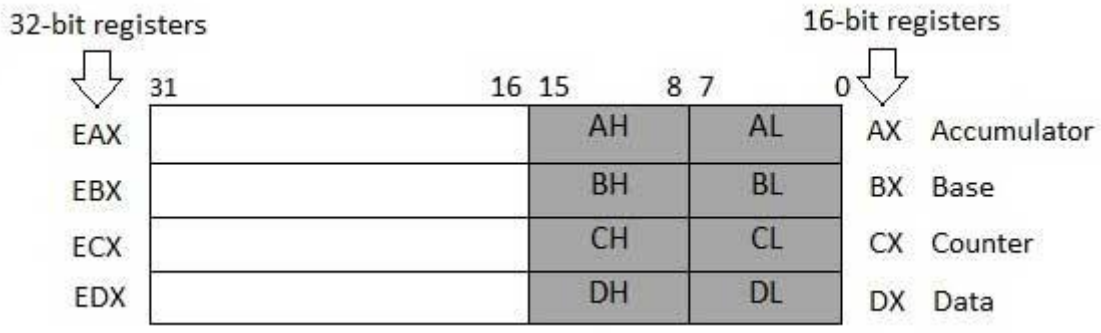
The general registers are further divided into the following groups −

- Data registers,
- Pointer registers, and
- Index registers.

Data Registers:

Four 32-bit data registers are used for arithmetic, logical, and other operations. These 32-bit registers can be used in three ways –

- As complete 32-bit data registers: EAX, EBX, ECX, EDX.
- Lower halves of the 32-bit registers can be used as four 16-bit data registers: AX, BX, CX and DX.
- Lower and higher halves of the above-mentioned four 16-bit registers can be used as eight 8-bit data registers: AH, AL, BH, BL, CH, CL, DH, and DL.



Some of these data registers have specific use in arithmetical operations.

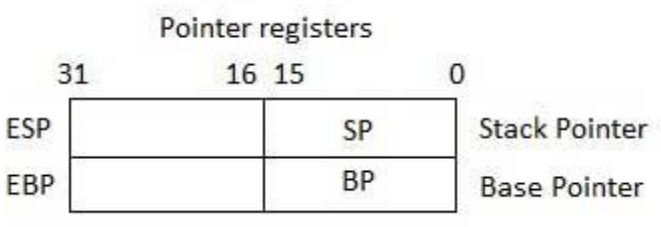
- **AX is the primary accumulator**; it is used in input/output and most arithmetic instructions. For example, in multiplication operation, one operand is stored in EAX or AX or AL register according to the size of the operand.

- **BX is known as the base register**, as it could be used in indexed addressing.
- **CX is known as the count register**, as the ECX, CX registers store the loop count in iterative operations.
- **DX is known as the data register**. It is also used in input/output operations. It is also used with AX register along with DX for multiply and divide operations involving large values.

Pointer Registers:

The pointer registers are 32-bit EIP, ESP, and EBP registers and corresponding 16-bit right portions IP, SP, and BP. There are three categories of pointer registers −

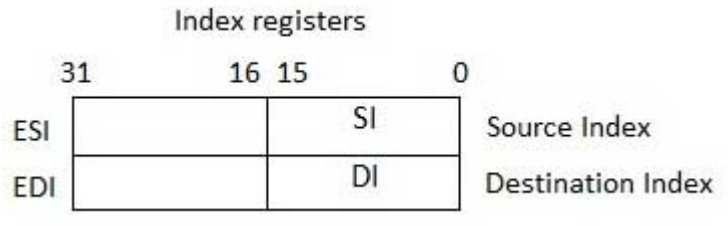
- **Instruction Pointer (IP)** − The 16-bit IP register stores the offset address of the next instruction to be executed. IP in association with the CS register (as CS:IP) gives the complete address of the current instruction in the code segment.
- **Stack Pointer (SP)** − The 16-bit SP register provides the offset value within the program stack. SP in association with the SS register (SS:SP) refers to be current position of data or address within the program stack.
- **Base Pointer (BP)** − The 16-bit BP register mainly helps in referencing the parameter variables passed to a subroutine. The address in SS register is combined with the offset in BP to get the location of the parameter. BP can also be combined with DI and SI as base register for special addressing.



Index Registers:

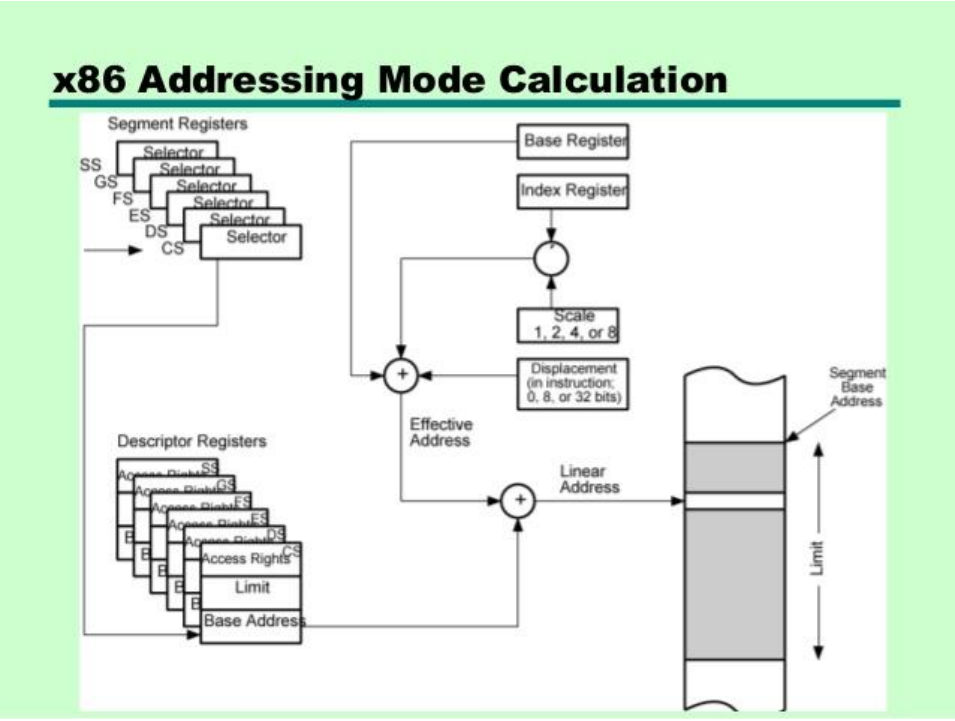
The 32-bit index registers, ESI and EDI, and their 16-bit rightmost portions. SI and DI, are used for indexed addressing and sometimes used in addition and subtraction. There are two sets of index pointers −

- **Source Index (SI)** − It is used as source index for string operations.
- **Destination Index (DI)** − It is used as destination index for string operations.



X86 ADDRESSING MODES:

- **Immediate mode:**
  The operand is included in the instruction. The operand can be a byte, word, or doubleword of data.
- **Register operand mode:**
  The operand is located in a register. For general instructions, such as data transfer, arithmetic, and logical instructions, the operand can be one of the 32-bit general registers (EAX,EBX, ECX, EDX, ESI, EDI, ESP, EBP), one of the 16-bit general registers (AX, BX, CX, DX, SI, DI,SP, BP), or one of the 8bit general registers (AH, BH, CH, DH, AL, BL, CL, DL). There are also some instructions that reference the segment selector registers (CS, DS, ES, SS, FS, GS).
- **Displacement mode:**
  The operand's offset (the effective address of figure) is contained as part of the instruction as an 8-, 16-, or 32-bit displacement. The displacement addressing mode is found on few machines because, as mentioned earlier, it leads to long instructions. In the case of the x86, the displacement value can be as long as 32 bits, making for a 6-byte instruction. Displacement addressing can be useful for referencing global variables. The remaining addressing modes are indirect, in the sense that the address portion of the instruction tells the processor where to look to find the address.
- **Base mode:**
  Specifies that one of the 8-, 16-, or 32-bit registers contains the effective address. This is equivalent to what we have referred to as register indirect addressing.
- **Base with displacement mode:**
  The instruction includes a displacement to be added to a base register, which may be any of the general-purpose registers.



Addressing mode calculation

- **Scaled index with displacement mode:**
  The instruction includes a displacement to be added to a register, in this case called an index register. The index register may be any of the general-purpose registers except the one called ESP, which is generally used for stack processing.
  In calculating the effective address, the contents of the index register are multiplied by a scaling factor of 1, 2, 4, or 8, and then added to a displacement. A scaling factor of 2 can be used for an array of 16-bit integers. A scaling factor of 4 can be used for 32-bit integers or floating-point numbers. Finally, a scaling factor of 8 can be used for an array of double-precision floating-point numbers.
- **Base with index and displacement mode:**
  Sums the contents of the base register, the index register, and a displacement to form the effective address. Again, the base register can be any general-purpose register and the index register can be any general-purpose register except ESP.
  This mode can also be used to support a two-dimensional array; in this case, the displacement points to the beginning of the array and each register handles one dimension of the array.
- **Based scaled index with displacement mode:**
  Sums the contents of the index register multiplied by a scaling factor, the contents of the base register, and the displacement. This is useful if an array is stored in a stack frame. This mode also provides efficient indexing of a two-dimensional array when the array elements are 2, 4, or 8 bytes in length.
- **Relative addressing:**
  Can be used in transfer-of-control instructions. A displacement is added to the value of the program counter, which points to the next instruction.

Mode	Algorithm
Immediate	Operand = A
Register Operand	LA = R
Displacement	LA = (SR) + A
Base	LA = (SR) + (B)
Base with Displacement	LA = (SR) + (B) + A
Scaled Index with Displacement	LA = (SR) + (I) × S + A
Base with Index and Displacement	LA = (SR) + (B) + (I) + A
Base with Scaled Index and Displacement	LA = (SR) + (I) × S + (B) + A
Relative	LA = (PC) + A

LA = linear address
(X) = contents of X
SR = segment register
PC = program counter
A = contents of an address field in the instruction
R = register
B = base register
I = index register
S = scaling factor

X86 Instruction set:

The 8086 microprocessor supports these types of instructions −

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- Program Execution Transfer Instructions (Branch & Loop 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.

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.