Lecture-2

CSE-231: MICROPROCESSOR & ASSEMBLY LANGUAGE

REFERENCES:

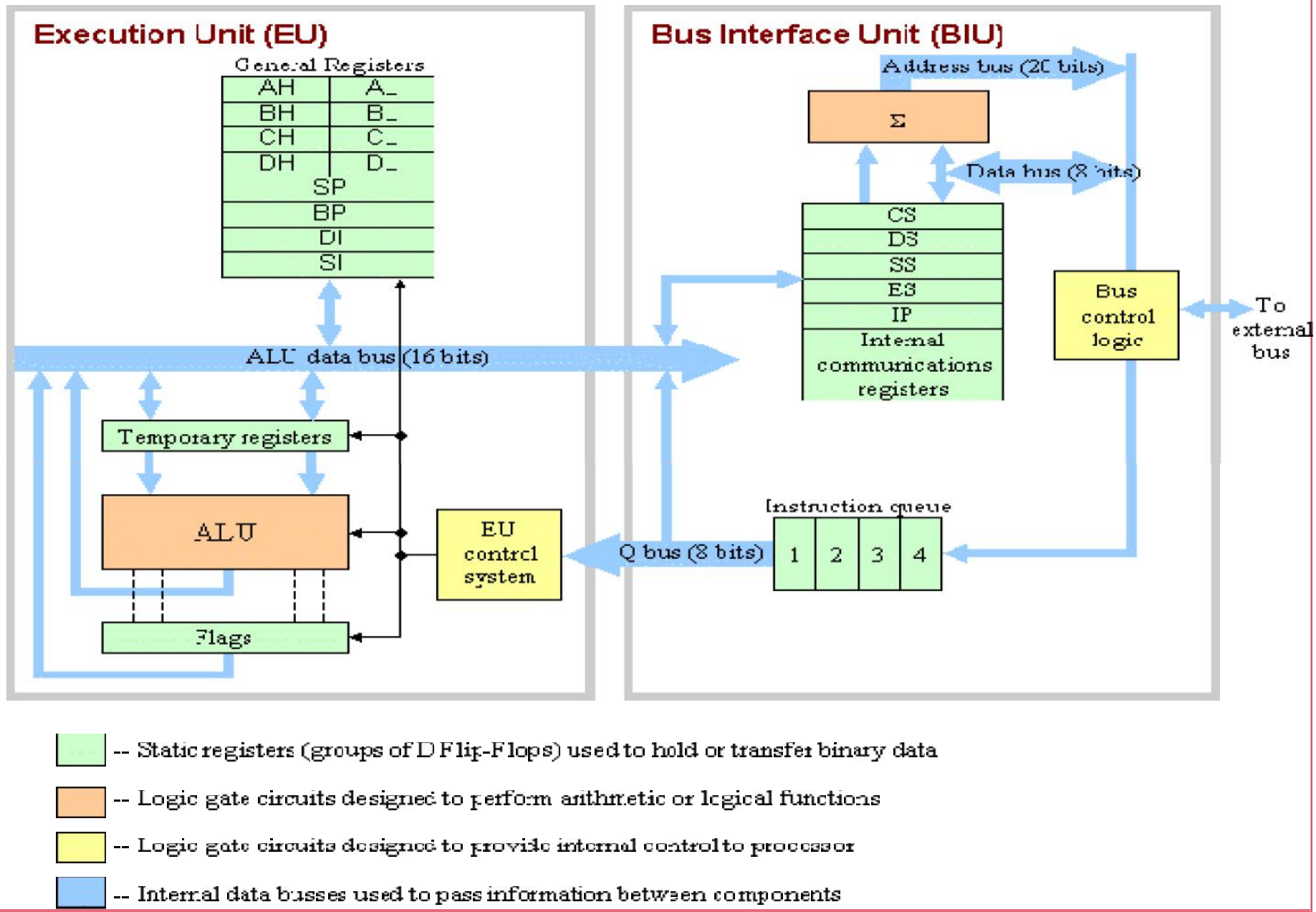
1. ASSEMBLY LANGUAGE PROGRAMMING AND ORGANIZATION OF THE IBM PC – CHARLES MARUT – CHAPTER 1(SEC 1.1.2,1.2), CHAPTER 3.
2. MICROPROCESSORS AND INTERFACING PROGRAMMING AND HARDWARE, SECOND EDITION, D.V. HALL – CHAPTER 2

Internal Architecture of 8086

The 8086 microprocessor is internally divided into two separate functional units.

- These are the Bus Interface Unit (BIU) and the Execution Unit (EU). The BIU fetches instructions, reads data from memory and ports, and writes data to memory and I/O ports. The EU executes instructions that have already been fetched by the BIU. The BIU and EU function independently.
- The BIU's instruction queue is a First-In First-out (FIFO) group of registers in which up to six bytes of instruction code are perfected from memory ahead of time.
- The BIU contains a dedicated adder, which is used to produce the 20-bit address.
- The bus control logic of the BIU generates all the bus control signals such as read and write signals for

  memory and I/O.


Execution Unit (EU)
General Registers
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
DI
SI
ALU data bus (16 bits)
Temporary registers
ALU
EU control system
Flags
Bus Interface Unit (BIU)
Address bus (20 bits)
Σ
Data bus (8 bits)
CS
DS
SS
ES
IP
Internal communications registers
Bus control logic
To external bus
Instruction queue
Q bus (8 bits)
1
2
3
4
-- Static registers (groups of D Flip-Flops) used to hold or transfer binary data
-- Logic gate circuits designed to perform arithmetic or logical functions
-- Logic gate circuits designed to provide internal control to processor
-- Internal data busses used to pass information between components

- The BIU has four 16-bit segment registers. These are the Code Segment (CS) register, the Data

  Segment (DS) register, the Stack Segment (SS) register, and the Extra Segment (ES) register.
- The BIU computes the 20-bit physical address internally using the programmer-provided logical address(16-bit contents of CS and IP) by logically shifting the contents of CS four bits to left and then adding the 16-bit contents of IP. For example, if [CS] = (456A)16 and [IP] = (1620)16, then the 20-bit physical address is generated by the BIU as follows:

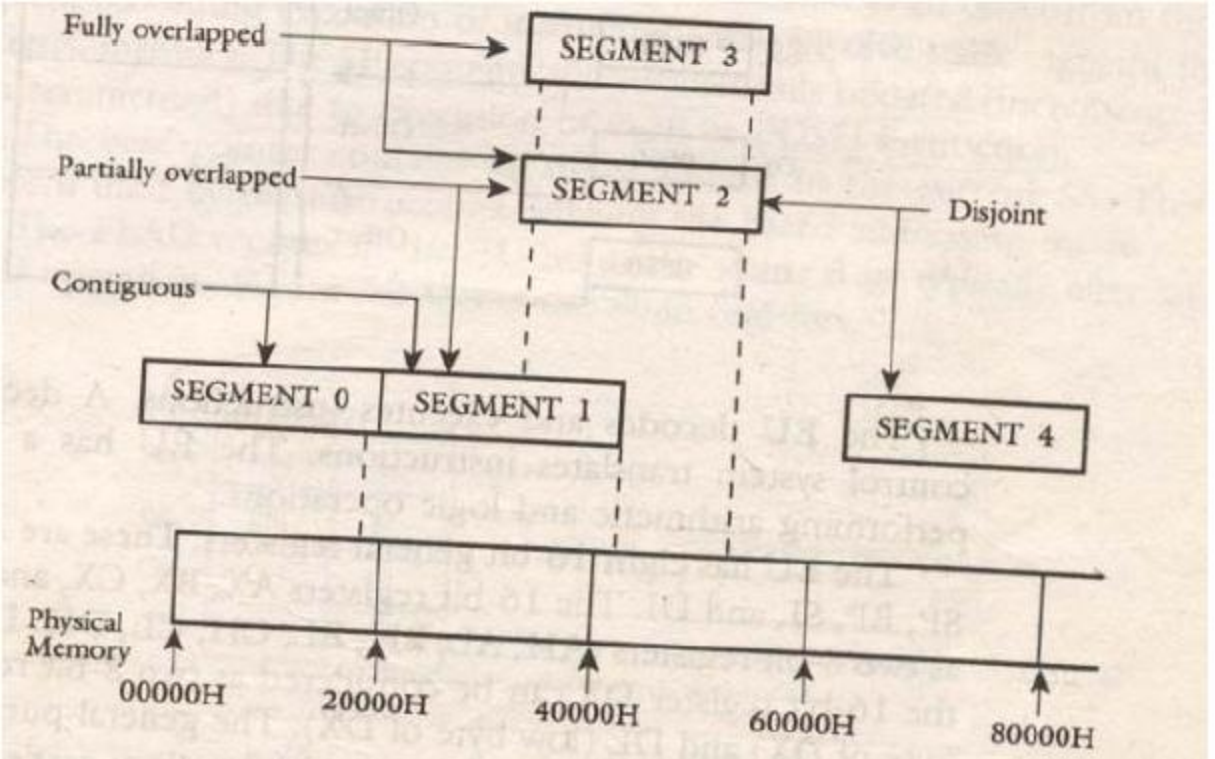
  Four times logically shifted [CS] to left = (456A0) 16

  + [IP] as offset = (1620)16

  20-bit physical address = (46CC0)16
- The SS register points to the current stack. The 20-bit physical stack address is calculated from SS and SP for stack instruction such as PUSH and POP.
- The DS register points to the current data segment; operands for most instructions are fetched from this segment. The 16-bit contents of Source Index (SI) or Destination Index (DI) are used as offset for computing the 20-bit physical address.
- The ES register points to the extra segment in which data (in excess of 64k pointed to by DS) is stored. String instructions always use ES and DI to determine the 20-bit physical address for the destination.

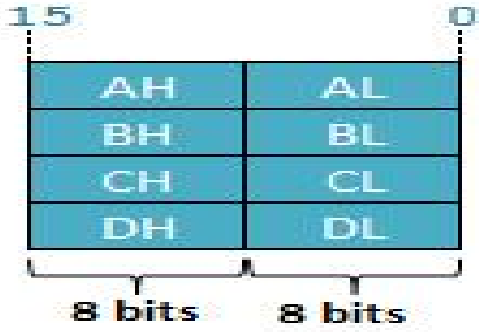
Segment address of 8086

- The segment can be continuous, partially overlapped, fully overlapped, or disjoint. An example of how five (segment 0 through segment 4) may be stored in physical memory are shown.



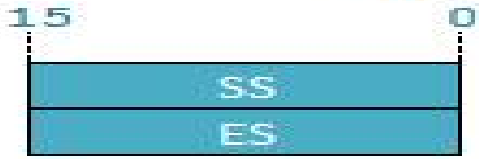
REGISTERS IN THE 8086 CPU

General Registers



AX (Primary accumulator)
BX (Base, accumulator)
CX (Counter, accumulator)
DX (Data, accumulator)

Segment Registers



Stack Segment
Extra Segment

Pointer & Index Registers

Base Pointer
Stack Pointer
Instruction Pointer

Status Register

Status Flags

GENERAL PURPOSE REGISTERS

- **AX - the accumulator register (divided into AH / AL):**
  1. Generates shortest machine code
  2. Arithmetic, logic and data transfer
  3. One number must be in AL or AX
  4. Multiplication & Division
  5. Input & Output
- **BX - the base address register (divided into BH / BL).**
- **CX - the count register (divided into CH / CL):**
  1. Iterative code segments using the LOOP instruction
  2. Repetitive operations on strings with the REP command
  3. Count (in CL) of bits to shift and rotate
- **DX - the data register (divided into DH / DL):**
  1. DX:AX concatenated into 32-bit register for some MUL and DIV operations
  2. Specifying ports in some IN and OUT operations

Pointer and Index Registers

- **SI - source index register:**
  1. Can be used for pointer addressing of data
  2. Used as source in some string processing instructions
  3. Offset address relative to DS
- **DI - destination index register:**
  1. Can be used for pointer addressing of data
  2. Used as destination in some string processing instructions
  3. Offset address relative to ES
- **BP - base pointer:**
  1. Primarily used to access parameters passed via the stack
  2. Offset address relative to SS
- **SP - stack pointer:**
  1. Always points to top item on the stack
  2. Offset address relative to SS
  3. Always points to word (byte at even address)
  4. An empty stack will had SP = FFFEh

SEGMENT REGISTERS

- **CS - points at the segment containing the current program.**
- **DS - generally points at segment where variables are defined.**
- **ES - extra segment register, it's up to a coder to define its usage.**
- **SS - points at the segment containing the stack.**

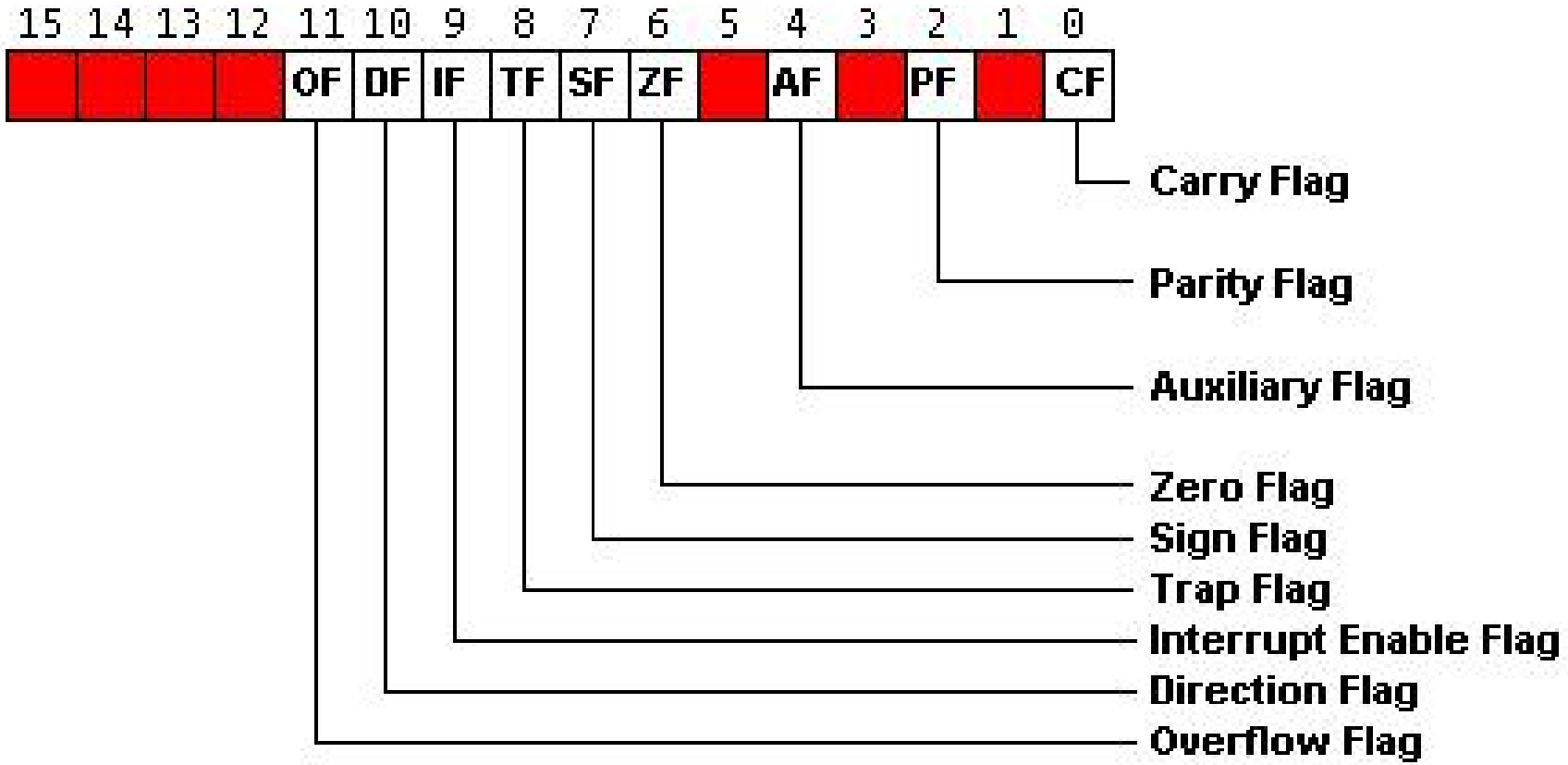
SPECIAL PURPOSE REGISTERS

- **IP - the instruction pointer:**
  1. Always points to next instruction to be executed
  2. Offset address relative to CS
- **IP register always works together with CS segment register and it points to currently executing instruction.**

Flag Register

- The 8086 have six one-bit flags.
  1. AF (Auxiliary carry flag) is used by BCD bit) into the high nibble or a borrow from the high nibble into

     the low nibble of the low-order 8-bit of a 16-bit number.
  2. CF (Carry Flag) is set if there is a carry from addition or borrow from subtraction.
  3. OF (Overflow Flag) is set if there is an arithmetic overflow, that is, if the size of the result exceeds the capacity of the destination location. An interrupt on overflow instructions is available which will generate an interrupt in this situation.
  4. SF (Sign Flag) is set if the most significant bit of the result is one (Negative) and is cleared to zero for non-negative result.
  5. PF (Parity Flag) is set if the result has even parity; PF is zero for odd parity of the result.
  6. ZF (Zero Flag) is set if the result is zero; ZF is zero for non-zero result.


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
OF DF IF TF SF ZF AF PF CF
Carry Flag
Parity Flag
Auxiliary Flag
Zero Flag
Sign Flag
Trap Flag
Interrupt Enable Flag
Direction Flag
Overflow Flag

- The 8086 has three control bits in the flag register which can be set or reset by the programmer:
  1. Setting DF (Direction Flag) to one causes string instructions to auto decrement and clearing DF to zero

     causes string instructions to auto increment.
  2. Setting IF (Interrupt Flag) to one causes the 8086 to recognize external mask able interrupts; clearing IF

     to zero disables these interrupts.
  3. Setting TF (Trace Flag) to one places the 8086 in the single-step mode. In this mode, the 8086 generate

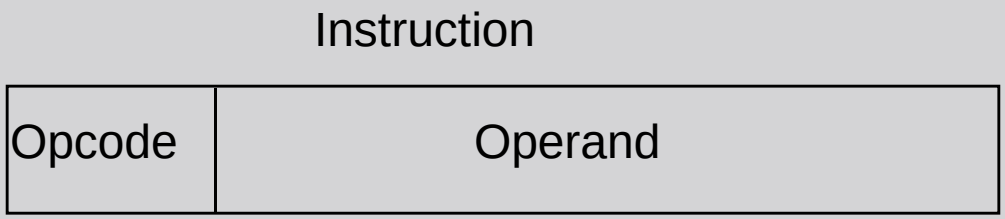
     an internal interrupt after execution of each instruction.

ADDRESSING MODES OF 8086

1. Immediate: In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

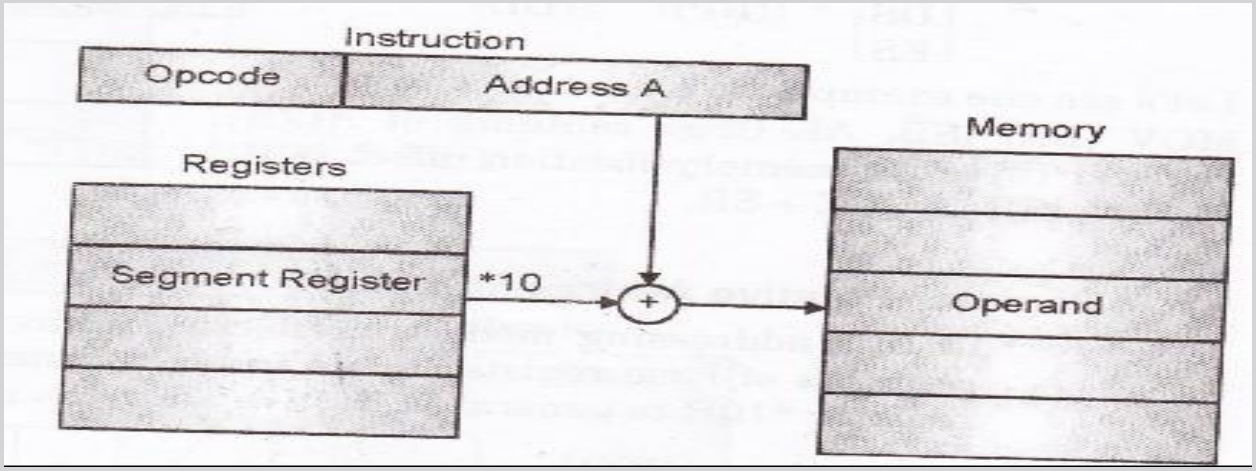
   Example: MOV AX, 0005H In the above example, 0005H is the immediate data.

   The immediate data may be 8-bit or 16-bit in size.



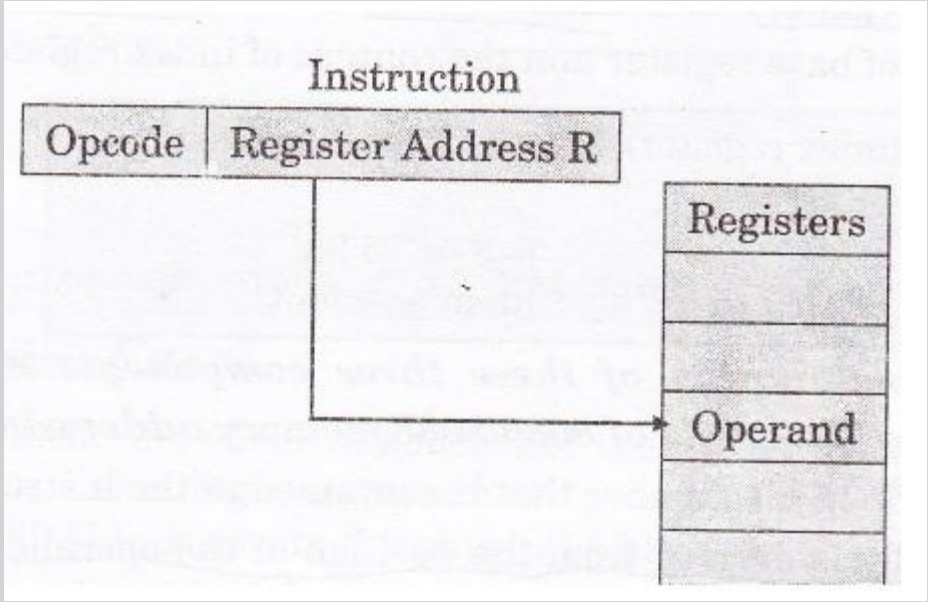
2. Direct: In the direct addressing mode, a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

Example: MOV AX, [5000H]. Here, data resides in a memory location in the data segment, whose effective address may be computed using 5000H as the offset address and content of DS as segment address. The effective address, here, is 10H*DS+5000H.



3. Register: In register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

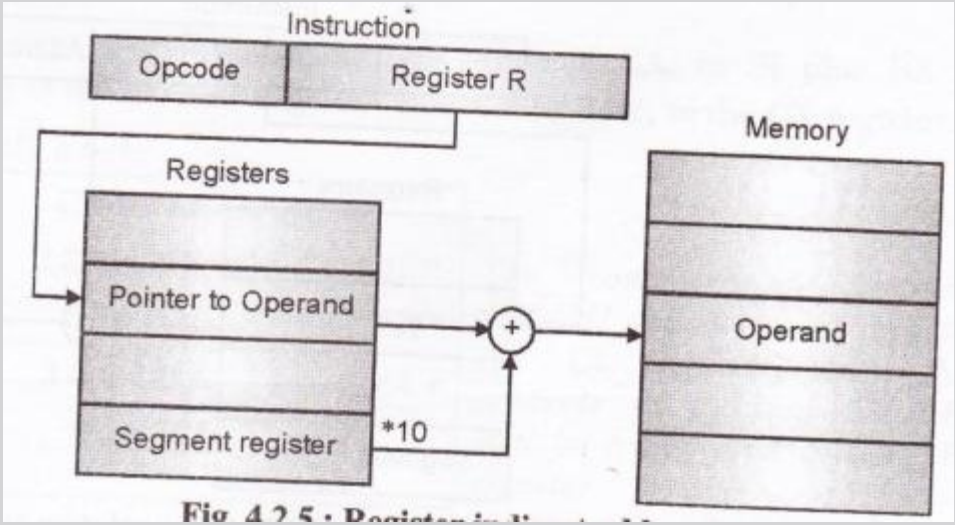
Example: MOV BX, AX.



4. Register Indirect: In this addressing mode, the offset address of data is in either BX or SI or DI registers.

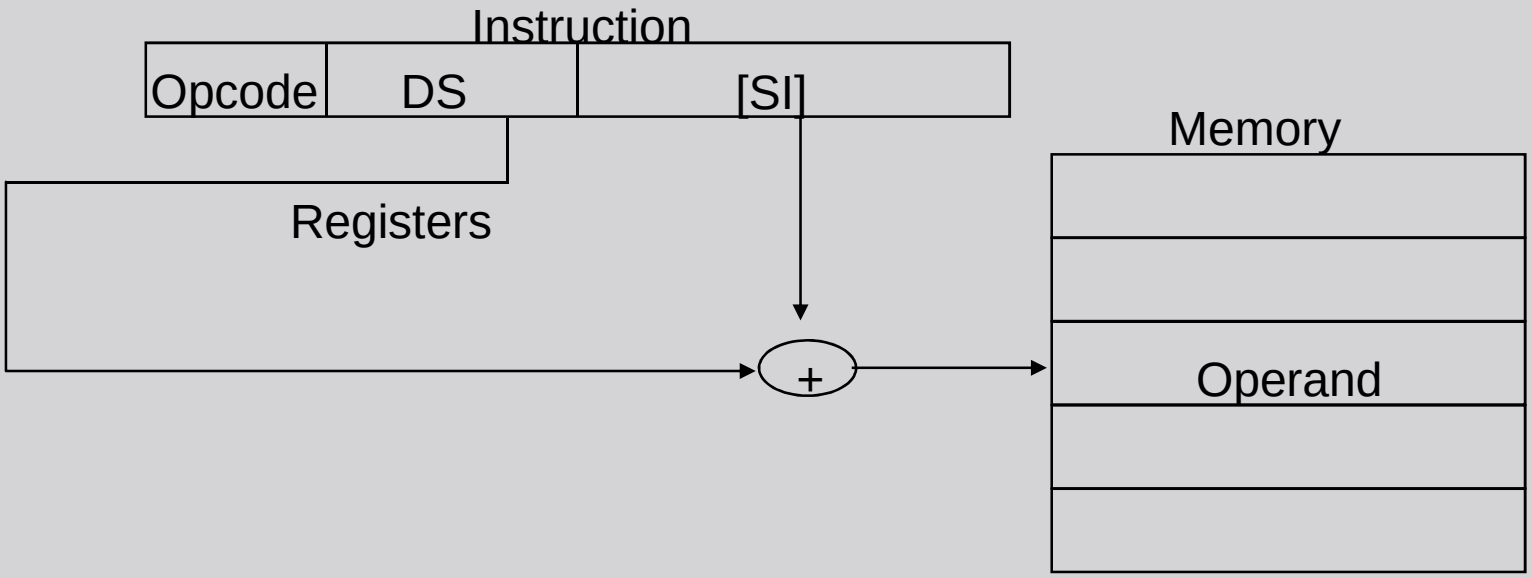
The default segment is either DS or ES.

Example: MOV AX, [BX]. Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as 10H*DS+ [BX].



5. Indexed: In this addressing mode, offset of the operand is stored in one of the index registers. DS and ES are the default segments for index registers SI and DI respectively. This mode is a special case of the above discussed register indirect addressing mode.

   Example: MOV AX, [SI]. Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as 10H*DS+ [SI].

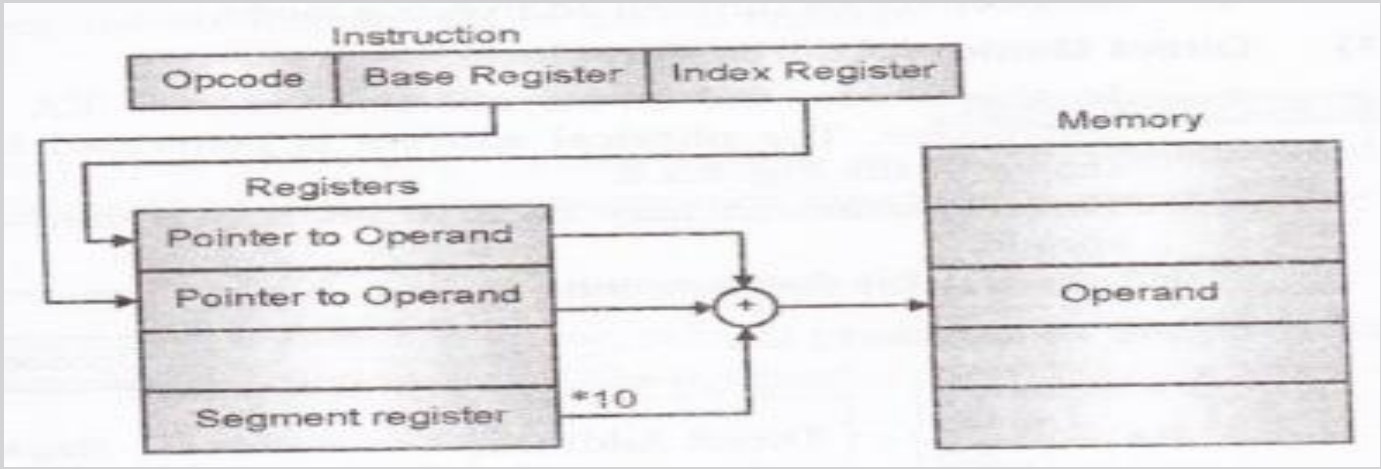


6. Register Relative: In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment. The example given before explains this mode.

   Example: MOV Ax, 50H [BX]. Here, effective address is given as 10H*DS+50H+ [BX].

7. Based Indexed: The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI).

Example: MOV AX, [BX] [SI]. Here, BX is the base register and SI is the index register. The effective address is computed as 10H*DS+ [BX] + [SI].



8. Relative Based Indexed: The effective address is formed by adding an 8-bit or 16-bit displacement with the sum of contents of any one of the bases registers (BX or BP) and any one of the index registers, in a default segment.

Example: MOV AX, 50H [BX] [SI]

Here, 50H is an immediate displacement, BX is a base register and SI is an index register. The effective address of data is computed as 10H*DS+ [BX] + [SI] + 50H.