(20 bit address bus, 16 bit data bus)

The 8086 architecture has been implemented using two-stage pipelining in instructions execution. The processor logic unit has been divided into **Bus Interface Unit (BIU)** and **Execution Unit (EU)**.

**Bus Interface Unit (BIU)** -

It carries out all bus operations for the EU, and it is responsible for executing all external bus cycles.

After fetched instructions codes in FIFO register, which is called a queue.

BIU fills the queue when the queue becomes empty spaces of two bytes. This process is known as pipeline flush.
Execution Unit (EU) -

The EU gets the opcode of an instruction from the instruction queue. Then the EU decodes and executes it. The BIU and EU operate independently.

The function of the EU is to execute all instructions, provide address to the BIU for fetching codes, and ef®ce and perform ALU operations after using various registers as well as the ®ag register.

2. Registers -
   (i) Data Register Group -
   AX Register - (16-bit register)
   BX Register - (16-bit register)
   CX Register - (16-bit register)
   DX Register - (16-bit register)

   AX, BX, CX, DX  - Used as an accumulator.

   Indirect register for MOV operation and base register while computing the data memory address.

   (ii) Segment Register - (16-bit register) → count register

   CS Register - (16-bit register) → Memory space = 64KB

   It is used for addressing a memory location in the code segment of the memory in which the program is stored for execution.

   DS Register -

   It points to the data segment of the memory, where data is stored.

   ES Register -

   It contains another extra data.

   SS Register -

   It points to the stack segment of the memory, where stack data is stored.

   (iii) Pointer and Index Register - (16-bit register)

   SP (Stack Pointer) - Used to locate the stack - top address.

   BP (Base Pointer) - Provide indirect access to data in a stack.
Source Index (SI) - to store offset and displacement. If the content of SI is added with the content of DS, to determine the physical address, it will be used as base address of data.

Destination Index (DI) - to store offset and displacement. If the content of DI is added with the content of ES to determine the physical address, it will be used as displacement address of data.

Instruction Pointer (IP) - Used as a program counter.

Flag Register - (16-bit Register)

Also called as Program Status Word (PSW). It has nine flags out of which nine are status flags (CF, PF, AF, ZF, SF, TF, OF) and there are control flags (TF, IF & OF).

<table>
<thead>
<tr>
<th>Flag</th>
<th>8 7</th>
<th>6 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>OF</td>
<td>DF</td>
<td>IF</td>
</tr>
<tr>
<td>SF</td>
<td>ZF</td>
<td>X</td>
</tr>
<tr>
<td>AF</td>
<td>X</td>
<td>PF</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>OF</td>
</tr>
</tbody>
</table>

- Carry Flag (CF) - 1, if carry is generated or borrow is generated, otherwise 0.
- Parking Flag (PF) - 1, if 8-bit operation contains even number of 1's, otherwise 0.
- Auxiliary Carry Flag (AF) - 1, if carry out of low to high nibble, otherwise 0.
- Zero Flag (ZF) - 1, if result of any ALU operation is zero, otherwise 0.
- Sign Flag (SF) - 0, if result of any ALU operation is positive number, otherwise 1.
- Overflow Flag (OF) - 1, if signed result cannot be expressed within the number of bits in the destination operand.

- Direction Flag (DF) - 1, string bytes can be accessed from a memory location address in document order, i.e. high memory address to low memory address, otherwise 0.
- Carry byte can be accessed from memory address in increasing order.

- Interrupt Enable Flag (IF) - 1, then if maskable interrupt in enabled, otherwise 0, if maskable interrupt are disabled.
- Trap Flag (TF) - 1, then a single step interrupt occurs after the current instruction executes and the program can be executed in single-step mode. The TF will be cleared by the single-step interrupt.
Physical Address = Segment Address + Offset Address

8086 Memory Address Range = 00000H - FFFFFFFH.

3. Memory Segmentation:
   - Segment without overlapping
   - Segment with overlapping

Advantages of segment memory:
(i) Allows memory capacity to be 1MB even though the addresses associated with the individual instructions are 16 bit wide.
(ii) Allows the use of separate memory areas like for code, data, stack etc.
(iii) Multitasking becomes easy.
(iv) Programs are relocatable so that programs can be run at any location in the memory.

4. 8086 Memory Addressing:

<table>
<thead>
<tr>
<th>Odd Bank</th>
<th>Even Bank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher Address Bank</td>
<td>Lower Address Bank</td>
</tr>
<tr>
<td>20009</td>
<td>20003</td>
</tr>
<tr>
<td>20007</td>
<td>20001</td>
</tr>
<tr>
<td>20005</td>
<td>20000</td>
</tr>
<tr>
<td>20004</td>
<td>20002</td>
</tr>
<tr>
<td>20002</td>
<td>20000</td>
</tr>
</tbody>
</table>

Address Bus: A15 - D7, Data Bus: D0 - D15, BHE, A0

<table>
<thead>
<tr>
<th>BHE</th>
<th>A0</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Both Banks active; 16 bit data transfer over banks on AD15 - AD0.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Only high bank active; one byte transfer on AD15 - AD0.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Only low bank active; one byte transfer on AD7 - AD0.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>No bank active.</td>
</tr>
</tbody>
</table>
Data can be accessed from memory in four different ways:

- 8-bit data from even-address range $A_0 = 0$, $BHE = 1$
- 8-bit data from odd-address range $A_0 = 1$, $BHE = 0$
- 16-bit data starting from even-address range $A_0 = 0$, $BHE = 0$
- 16-bit data starting from odd-address range $A_0 = 1$, $BHE = 1$

2nd Cycle $A_0 = 1$, $BHE = 0$

2. Memory Read and Write Bus Cycle of 8086

When any external memory or I/O devices are accessed, only four clock cycles are required to perform a read or write operation. These four clock cycles are grouped, which is called bus cycle.

Memory Read Bus Cycles for Minimum mode:

- **Address:** $A_{15}-A_0$
- **Data from memory:** $D_{15}-D_0$
- **Status:** $S_6-S_0$
- **BHE:** $BHE = 1$ for 16-bit
- **BHE:** $BHE = 0$ for 8-bit
Demultiplexing of the system has in 8086 and 8088 microprocessors:

- 74LS373 \rightarrow 8 IC's
- 74LS245 \rightarrow 8 Buffers
- 74LS244 \rightarrow 8 IC's or Buffers

In 8086, 2 IC's are fully utilized and one back is parasitically used.
Demultiplexing of fully buffered system bus in 8086 processors
Addressing modes of 8086:

(i) Immediate Addressing - load data immediately. Eg: MOV BX, 7000H

(ii) Register Addressing - source register to destination register. Eg: MOV AL, 81

(iii) Memory Addressing -

16-bit Effective address = Base + Index + Displacement

20-bit physical address = Segment x 10 + Base + Index + Displacement

Direct Addressing -

Specify memory address (effective address) where data is stored. Eg: MOV AX, [5000H]  [DS x 10 + 5000H]

Register Indirect Addressing -

Specify a register containing an address (effective address), where data is stored. Eg: MOV AL, [BX]

Base Indirect Addressing -

The 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to the location where the data resides. Eg: MOV AL, [BX + 8-WR DISP]

Indirect Addressing -

The 8-bit or 16-bit instruction operand is added to the contents of an indirect register (SI or DI), the resulting value is a pointer to the location where the data resides. Eg: MOV AL, CS: [SI + DISP]

Base Indirect with Displacement Addressing -

The contents of a base register (BX or BP) is added to the contents of an indirect register (SI or DI), the resulting value is a pointer to the location where the data resides. Eg: MOV AL, [BX + DI]

Base Indirect with Displacement Addressing -

The 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and indirect register (SI or DI), the resulting value is a pointer to the location where the data resides. Eg: MOV AL, [BX + DI + DISP]

String Addressing mode -

DS: SI is used as a source of string and ES: DI is used to locate the destination address of the string. Eg: MOV SB (move string from source to destination)
(i) Branch addressing:

- **Intrasegment Direct**: 
  \[ EA = \text{content of } \text{IP} + 8 + \text{16-bit displacement} \]

- **Intrasegment Indirect**: 
  \[ EA = \text{content of a register-memory} + 8 + \text{16-bit displacement (effective branch address)} \]

- **Intrasegment Direct**: 
  This replaces the content of IP with a part of the instruction and the content of CS with another part of the instruction.

- **Intrasegment Indirect**: 
  This makes the content of IP and CS with the content of two consecutive words in memory that are referenced using any of the above data-related addressing modes except the immediate and register mode.

(b) Instruction set of 8086:

i) Data Transfer Instructions: `MOV, XCHG, LAHF` (load the lower flags byte into AH), `SAHF` (Save AH into lower flag byte), `IN, OUT, LEA` (load effective address), `LDS` (load data segment), `LES` (load extra segment), `XLAT` (Translate byte in AL by table look-up), `PUSH, POP, PUSHF` (Push flag word onto stack), `POPF` (Pop word at top of stack to flags registers).

ii) Arithmetic and Logical Instructions: `ADD, SUB, INC, DEC, CMP, ADC` (add two operands with carry from previous add), `SBB` (subtract source and the carry flag bit from destination), `NEC` (change the sign of an operand), `MUL, IMUL` (signed multiplication), `DIV, IDIV` (signed division), `DAA` (Decimal adjustment after addition), `DAS` (Decimal adjustment after subtraction), `AAA` (ASCII adjust for addition), `AAS` (ASCII adjust for subtraction), `AAM` (ASCII adjustment for multiplication), `CBW` (Convert from byte to word), `CWD` (Convert from word to double word).

Logical: `NOT, AND, OR, XOR, TEST` (Non-destructive logical AND).
Shift and Rotate Instructions - 
SHL/SAL (Shift logical/arithmetic left), SHR (shift logical right), SAR (shift arithmetic right), ROH (Rotate right without carry), ROL (Rotate left without carry), RCR (Rotate right through carry), RCL (Rotate left through carry).

Branch Instructions - JMP, JNZ (Jump if zero), JNE (Jump if not equal), JEQ (Jump if equal), JGE (Jump if greater or equal), JNG (Jump if not greater), JG (Jump if greater), JNL (Jump if not less), JNLE (Jump if not less or equal), JNO (Jump if not overflow), JNP (Jump if not parity), JNS (Jump if not sign).

Loop Instructions - LOOP (loop to short target), LOOPNZ (loop to short target if Z bit is clear), LOOPNE/NZ (loop to short target if Z bit is clear).

CALL (call a procedure), RET (return from procedure), INT, INTO (interrupt on overflow), IRET (return from interrupt without content).

String Instructions - MOVSW (move string word), MOVSB (move string byte), STOSB (store string byte), STOSW (store string word), LODSW (load string word), LODSB (load string byte), CMPSB (compare string byte), CMPSW (compare string word), SCASB (scan string byte), SCASW (scan string word), REPEAT Instruction, REP/REP/EPE/REPZ (repeat string instruction), REPZ/REPNE (repeat instruction while not zero).

Processor Control Instructions - CLC (clear the carry flag), CMC (complement the carry flag), SIF (set the carry flag), CLD (clear direction flag), STD (set direction flag), CLI (clear interrupt flag), STI (set interrupt flag),HLT, WAIT, LOCK (lock bus), NOP (no operation), ESC (escape).

Flag manipulation instruction - CLD, STD, CLI, STI etc.

Assembly language Programs of 8086 microprocessor.