lecture-3
Programming with 8086 microprocessor

Internal Architecture and Features of 8086 Microprocessor

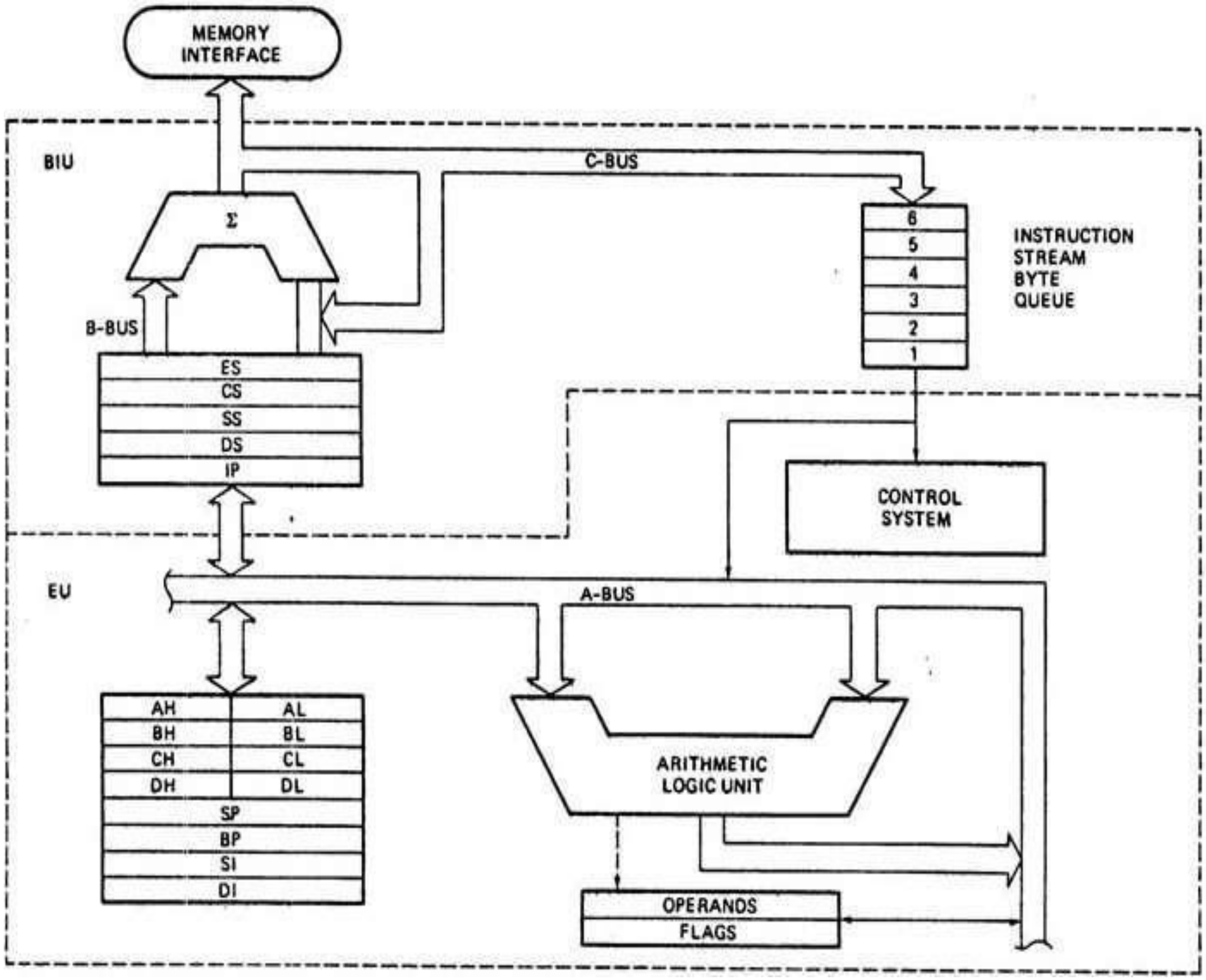

Fig: Internal Block Diagram of 8086 Microprocessor

Features of 8086 microprocessor

- Intel 8086 is a widely used 16 bit microprocessor.
- The 8086 can directly address 1MB of memory.
- The internal architecture of the 8086 microprocessor is an example of register based microprocessor and it uses segmented memory.
- It pre-fetches up to 6 instruction bytes from the memory and queues them in order to speed up the instruction execution.

- It has data bus of width 16 bits and address bus of width 20 bits. So it always accesses a 16 bit word to or from memory.
- The 8086 microprocessor is divided internally into two separate units which are Bus interface unit (BIU) and the execution unit (EU).
- The BIU fetches instructions, reads operands and write results.
- The EU executes instructions that have already been fetched by BIU so that instructions fetch overlaps with execution.
- A 16 bit ALU in the EU maintains the MP status and control flags, manipulates general register and instruction operands.

Bus Interface Unit(BIU) and its Components

The BIU sends out addresses, fetches instructions from memory reads data from memory or ports and writes data to memory or ports. So it handles all transfers of data and address on the buses for EU. It has main 2 parts instruction queue and segment registers.

- The BIU can store up to 6 bytes of instructions with FIFO (First in First Out) manner in a register set called a queue. When EU is ready for next instruction, it simply reads the instruction from the queue in the BIU. This is done in order to speed up program execution by overlapping instruction fetch with execution. This mechanism is known as pipelining.
- The BIU contains a dedicated address, which is used to produce 20 bit address. Four segment registers in the BIU are used to hold the upper 16 bits of the starting address of four memory segments that the 8086 is working at a particular time. These are code segment, data segment, stack segment and extra segment. The 8086's 1 MB memory is divided into segments up to 64KB each.

- **Code segment register and instruction pointer (IP):** The CS contains the base or start of the current code segment. The IP contains the distance or offset from this address to the next instruction byte to be fetched. Code segment address plus an offset value in the IP indicates the address of an instruction to be fetched for execution.

- **Data Segment**

  Data segment Contains the starting address of a program's data segment. Instructions use this address to locate data. This address plus an offset value in an instruction, causes a reference to a specific byte location in the data segment.

- **Stack segment (SS) and Stack Pointer (SP)**

  Stack segment Contains the starting address of a program's stack segment. This segment address plus an offset value in the stack pointer indicates the current word in the stack being addressed.

- **Extra Segment(ES)**

  It is used by some string (character data) to handle memory addressing. The string instructions always use the ES and destination index (DI) to determine 20 bit physical address.

  Execution Unit (EU)

  The EU decodes and executes the instructions. The EU contains arithmetic and logic (ALU), a control unit, and a number of registers. These features provide for execution of instructions and arithmetic and logical operations. It has nine 16 bit registers which are AX, BX, CX, DX, SP, BP, SI, DI and flag. First four can be used as 8 bit register (AH, Al, BH, BL, CH, DH, DL)

- **AX Register**

  AX register is called 16 bit accumulator and AL is called 8 bit accumulator. The I/O (IN or OUT) instructions always use the AX or AL for inputting/Outputting 16 or 8 bit data from or to I/O port.

- **BX Register**

  BX is known as the base register since it is the only general purpose register that can be used as an index to extend addressing. The BX register is similar to the 8085's H, L register. BX can also be combined with DI or SI as C base register for special addressing.

- **CX register:**

  The CX register is known as the counter register because some instructions such as SHIFT, ROTATE and LOOP use the contents of CX as a Counter.

- **DX register:**

  The DX register is known as data register. Some I/O operations require its use and multiply and divide operations that involve large values assume the use of DX and AX together as a pair. DX comprises the rightmost 16 bits of the 32-bit EDX.

- **Stack Pointer (SP) and Base Pointer (BP):**

  Both are used to access data in the stack segment. The SP is used as an offset from the current stack segment during execution of instructions. The SP's contents are automatically updated (increment/decrement) during execution of a POP and PUSH instructions.

  The BP contains the offset address in the current stack segment. This offset is used by instructions utilizing the based addressing mode.

- **Index register:**

  The two index registers SI (Source index) and DI (Destination Index) are used in indexed addressing. The instructions that process data strings use the SI and DI index register together with DS and ES respectively, in order to distinguish between the source and destination address.

- **Flag register:**

  The 8086 has nine 1 bit flags. Out of 9 six are status and three are control flags. The control bits in the flag register can be set or reset by the programmer.

				O	D	I	T	S	Z		A		P		C

D_{15} D_0

- **O- Overflow flag** This flag is set if an arithmetic overflow occurs, i.e. if the result of a signed operation is large enough to be accommodated in a destination register.
- **D-Direction Flag** This is used by string manipulation instructions. If this flag bit is '0' , the string is processed beginning from the lowest address to the higher address, i.e. auto incrementing mode otherwise the string is processed from the highest address towards the lowest address, i.e. autodecrementing mode.
- **I-Interrupt flag** If this flag is set the maskable interrupts are recognized by the CPU, otherwise they are ignored.
- **T- Trap flag** If this flag is set the processor enters the single step execution mode. In other words, a trap interrupt is generated after execution of each instruction. The processor executes the current instruction and the control is transferred to the Trap interrupt service routine.
- **S - Sign flag:** This flag is set when the result of any computation is negative. For signed computations, the sign flag equals the MSB of the result.
- **Z- Zero** This flag is set when the result of the computation is or comparison performed by the previous instruction is zero. 1 for zero result, 0 fir nonzero result
- **A- Auxiliary Carry** This is set if there is a carry from the lowest nibble, i.e. bit three during the addition or borrow for the lowest nibble i.e. bit three, during subtraction.
- **P- Parity flag** This flag is set to 1 if the lower byte of the result contains even number of 1s otherwise reset.
- **C-Carry flag** This flag is set when there is a carry out of MSB in case of addition or a borrow in case of subtraction.

SEGMENT AND OFFSET ADDRESS:

- Segments are special areas defined in a program for containing the code, data and stack. A segment begins on a paragraph boundary. A segment register is of 16 bits in size and contains the starting address of a segment.
- A segment begins on a paragraph boundary, which is an address divisible by decimal 16 or hex 10. Consider a DS that begins at location 038EOH. In all cases, the rightmost hex digit is zero, the computer designers decided that it would be unnecessary to store the zero the zero digit in the segment register. Thus 038E0H is stores in register as 038EH.
- The distance in bytes from the segment address to another location within the segment is expressed as an offset or displacement. Suppose the offset of 0032H for above example of data segment. Processor combines the address of the data segment with the offset as:

- **SA: OA** (segment address: offset address) 038E: 0032 H = 038E * 10 +0032
  = 038EO + 0032
  Physical address = 03912H

<u>**Instructions in 8086**</u>

1) Arithmetic Instructions

a) ADD reg_8 /mem_8 , reg_8/mem_8/ $Immediate_8$

ADD reg_{16}/mem_{16} , reg_{16}/ mem_{16}/ $Immediate_{16}$
E.g. ADD AH, 15 ; It adds binary number
ADD AH, NUM1 ADD Al, [BX]
ADD [BX], CH/CX ADD AX,[BX]

b) **ADC**: Addition with Carry
ADC reg/ mem, reg/mem/Immediate data

c) **SUB:** Subtract 8 bit or 16 bit binary numbers
SUB reg/mem, reg/mem/Immediate

d) **SBB:** Subtract with borrow
SBB reg/mem, reg/mem/Immediate

e) **MUL** : unsigned multiplication
MUL reg_8/mem_8 (8 bit accumulator – AL)
MUL reg_{16}/ mem_{16} (16 bit accumulator-Ax)
E.g. MUL R_8 $(multiplier) \equiv R_8 \times AL \rightarrow$ AX (16 bit result)
MUL R_{16} $(multiplier) \equiv R_{16} \times AL \rightarrow$ DX:AX (32 bit result)
IMUL – signed multiplication
Same operation as MUL but takes sign into account

f) **DIV** reg/mem
E.g. DIV $R_8 \equiv AX/R_8 \rightarrow$ (Remainder $\rightarrow$ AH) & (Q$uotient \rightarrow$ AL)
DIV $R_{16} \equiv DX:AX/R_{16} \rightarrow$ (R$emainder \rightarrow$ DX) & (Q $uotient \rightarrow$ AX)
IDIV- Signed division
Same operation as DIV but takes sign into account.

g) **INC/DEC** (Increment/Decrement by
1) INC/DEC reg./mem. (8 bit or 16bit)
E.g. INC AL DEC BX
INC NUM1

h) **NEG**- Negate (2's complement)

i) **ASCII-BCD** Conversion

AAA: ASCII Adjust after addition
AAS: ASCII Adjust after subtraction
AAM: Adjust after multiplication
AAD: Adjust after division
DAA: Decimal adjust after addition
DAS: Decimal adjust after subtraction

2) Logical/shifting/comparison instructions

a) Logical

AND/OR/XOR reg/mem, reg/mem/immediate NOT reg/mem

```
E. g.  AND  AL,  AH
       XOR [BX], CL
```

b) Rotation

ROL- rotate left, ROR-rotate right

```
E.g.    ROL AX, 1        ; rotated by 1
        ROL AX, CL       ; if we need to rotate more than one bit
```

RCL-rotate left through carry

RCR-rotate right through carry

```
E.g.    RCL AX, 1
        RCL AX, CL       ; Only CL can be used
```

c) Shifting

SHL -logical shift left

SHR - logical shift right

Shifts bit in true direction and fills zero in vacant place

E.g. SHL reg/mem, 1/CL

arithmetic shift left

SAR- arithmetic shift right

Shifts bit/word in true direction, in former case place zero in vacant place and in later case place previous sign in vacant place.

E.g. 1 011010 [1 $\Longrightarrow$ 11011010

d) Comparison

CMP –compare

CMP reg/mem, reg/mem/immediate

E.g. CMP BH, AL

Operand1		Operand 2	CF	SF	ZF
	>		0	0	0
	=		0	0	1
	<		1	1	0

TEST: test bits (AND operation)

TEST reg/mem, reg/mem/immediate

3) Data Transfer Instructions:

MOV reg./mem , reg./mem./immediate
LDS: Load data segment register
LEA: load effective address
LES: Load extra segment register
LSS: Load stack segment register
E.g. LEA BX, ARR = MOV BX, OFFSET ARR
LDS BX, NUM1

Segment address $\rightarrow$ DS
Offset address $\rightarrow$ BX

XCHG reg/mem, reg/mem
E.g. XCHG AX, BX
XCHG AL, BL
XCHG CL,[BX]
IN AL, DX ; DX: Port address, AH also in AL
OUT DX, AL/AH

4) Flag Operation

CLC: Clear carry flag
CLD: Clear direction flag
CLI: Clear interrupt flag
STC: Set Carry flag
STD: Set direction flag
STI: Set Interrupt flag
CMC: Complement Carry flag
LAHF: Load AH from flags (lower byte) SAHF: Store AH to flags
PUSHF: Push flags into stack
POPF: Pop flags off stack

5) STACK Operations

PUSH reg_{16}

POP reg_{16}

6) Looping instruction (CX is automatically used as a counter)

LOOP: loop until complete
LOOPE: Loop while equal
LOOPZ: loop while zero
LOOPNE: loop while not equal
LOOPNZ: loop while not zero

7) Branching instruction

a) Conditional

JA: Jump if Above
JAE: Jump if above/equal
JB: Jump if below
JBE: Jump if below/equal
JC: Jump if carry
JNC: Jump if no carry
JE: Jump if equal JNE:
Jump if no equal JZ:
Jump if zero
JNZ: Jump if no zero
JG: Jump if greater
JNG: Jump if no greater
JL: Jump if less
JNL: Jump if no less
JO: jump if overflow
JS: Jump if sign JNS:
Jump if no sign JP:
jump if plus
JPE: Jump if parity even
JNP: Jump if no parity
JPO: Jump if parity odd

b) Unconditional

CALL: call a procedure	RET: Return
INT: Interrupt	IRET: interrupt return
JMP: Unconditional Jump	RETN/RETF: Return near/Far

8) Type conversion

CBW: Convert byte to word
CWD: Convert word to double word

AX → DX: AX

9) <u>String instructions</u>

a)	MOVS/ MOVSB/MOVSW	; Move string
	DS: SI source	
	DS: DI destination	
	CX: String length	
b)	CMPS/ CMPSB/CMPW	; Compare string
c)	LODS /LODSB/LODW	; Load string
d)	REP	; Repeat string

ng and DI is used to point to the first byte or word of the destination string, when string instruction is executed. The SI or DI is automatically incremented or decremented to point to the next byte or word depending on the direction flag (DF).
E.g. MOVS, MOVSB, MOVSW

<u>Examples:</u>

```
TITLE Program to add ten numbers
        .MODEL SMALL
        .STACK 64
        .DATA
ARR DB 73, 91, 12, 15, 79, 94, 55,
89 SUM DW ?
        .CODE
MAIN PROC FAR
    MOV AX, @DATA
    MOV DS, AX
```

```
      MOV CX, 10
      MOV AX, 0
      LEA BX, ARR
L2: ADD Al, [BX]
      JNC L1
      INC AH
L1: INC BX
      LOOP L2
      MOV SUM, AX
      MOV AX, 4C00H
      INT 21H
MAIN ENDP
END MAIN
```