

COMPUTER ORGANIZATION LAB RECORD FRONT PAGES

8086 INTRODUCTION:

8086 Microprocessor is an enhanced version of 8085 Microprocessor that was designed by Intel in 1976. It is a 16-bit Microprocessor having 20 address lines and 16 data lines that provides up to 1MB storage. It consists of powerful instruction set, which provides operations like multiplication and division easily.

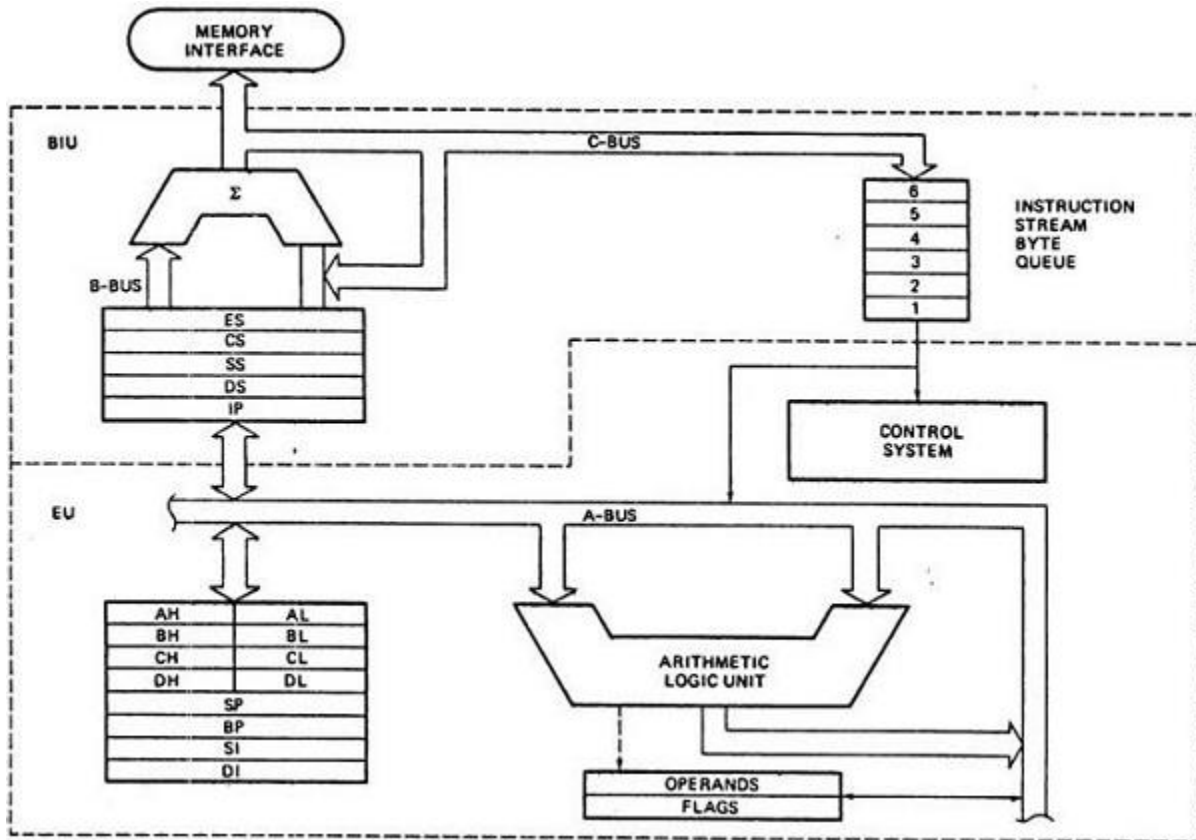
It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

Features of 8086

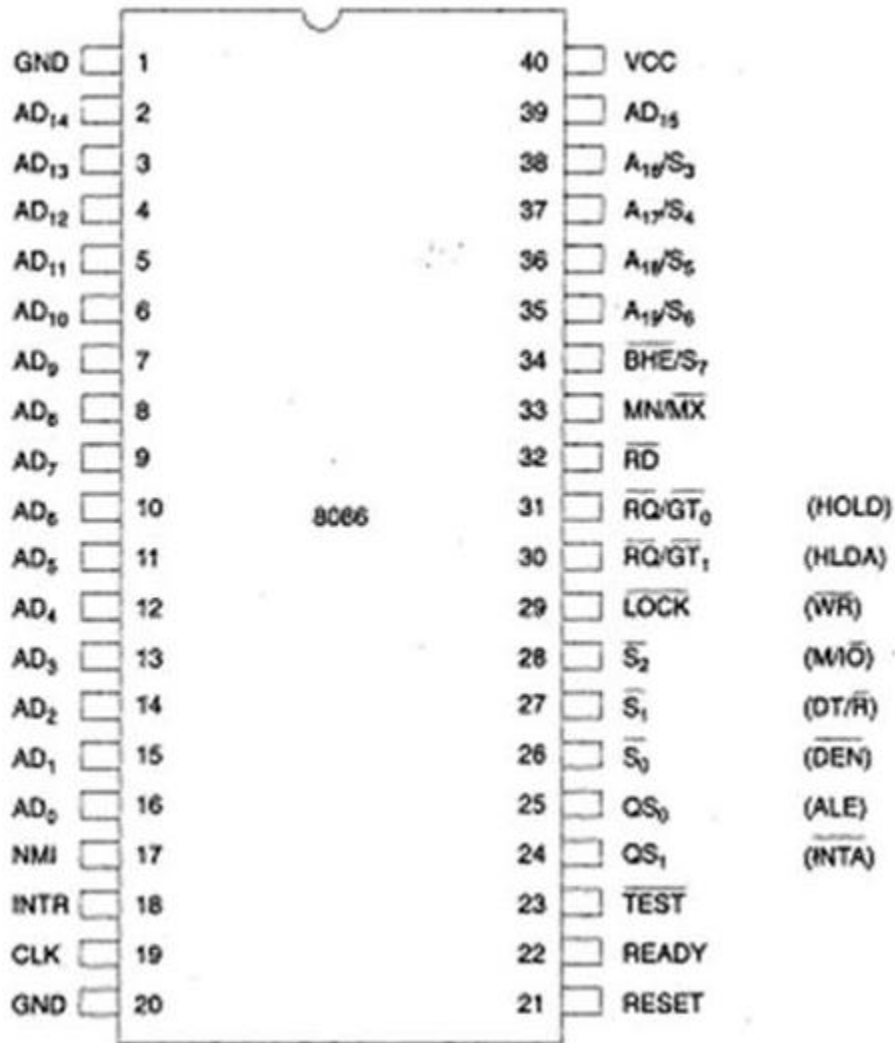
The most prominent features of a 8086 microprocessor are as follows –

- It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing.
- It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
- It is available in 3 versions based on the frequency of operation –
 - 8086 → 5MHz
 - 8086-2 → 8MHz
 - (c)8086-1 → 10 MHz
- It uses two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance.
- Fetch stage can prefetch up to 6 bytes of instructions and stores them in the queue.
- Execute stage executes these instructions.
- It has 256 vectored interrupts.
- It consists of 29,000 transistors.

Architecture of 8086



8086 Pin Diagram



8086 INSTRUCTION SET

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

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 $CX \neq 0$.
- **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/MOVS/BS/MOVS** – Used to move the byte/word from one string to another.
- **COMS/COMPS/BS/COMPS** – Used to compare two string bytes/words.
- **INS/INS/BS/INS** – Used as an input string/byte/word from the I/O port to the provided memory location.
- **OUTS/OUTS/BS/OUTS** – Used as an output string/byte/word from the provided memory location to the I/O port.
- **SCAS/SCAS/BS/SCAS** – Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
- **LODS/LODS/BS/LODS** – 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

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]
```

LIST OF EXP

ARTHEMATIC OPERATIONS

1. write an assembly language program(ALP) to find Addition of 2-16 bit numbers using 8086 trainer kit?
2. write an assembly language program(ALP) to find subtraction of 2-16 bit numbers using 8086 trainer kit?
3. write an assembly language program(ALP) to find multiplication of 2-16 bit numbers using 8086 trainer kit?
4. write an assembly language program(ALP) to find division of 2-16 bit numbers using 8086 trainer kit?
5. write an assembly language program(ALP) to find Addition of 2-8 bit numbers using 8086 trainer kit?
6. write an assembly language program(ALP) to find subtraction of 2-8 bit numbers using 8086 trainer kit?
7. write an assembly language program(ALP) to find multiplication of 2-8 bit numbers using 8086 trainer kit?
8. write an assembly language program(ALP) to find division of 2-8 bit numbers using 8086 trainer kit?

ADDRESSING MODES

9. write an assembly language program(ALP) to perform immediate addressing mode
10. write an assembly language program(ALP) to perform Direct addressing mode
11. write an assembly language program(ALP) to perform Register addressing mode
12. write an assembly language program(ALP) to perform indirect addressing mode
13. write an assembly language program(ALP) to perform based-indexed addressing mode

BRANCH INSTRUCTION

14. write an assembly language program(ALP) to find largest byte from given 2-Bytes

SORTING

15. write an assembly language program(ALP) to arrange given numbers in ascending order
16. write an assembly language program(ALP) to arrange given numbers in descending order

EXECUTION PROCEDURE ON 8086 TRAINER KIT

START

switch on power supply and check the display with MP:86

↓
A 2000 (ENTER)

↓
able to see memory address starting from 0000:2000

↓
Type Program till the last instruction i.e INT 03

↓
**then Type (SHIFT+!)
(if direct input in register then type G 2000)**

OR

↓
INPUT

↓
S ADDR(2100) if memory locations

OR

X REGNAME(AX) for register

↓
EXECUTION

G 2000

↓
OUTPUT

S ADDR(2100) hit enter

OR

X REGNAME

↓
STOP

Note: if you want to enter more than 1 input(S 2100) press (,) for multiple values till we finish the data and press enter after giving last value.