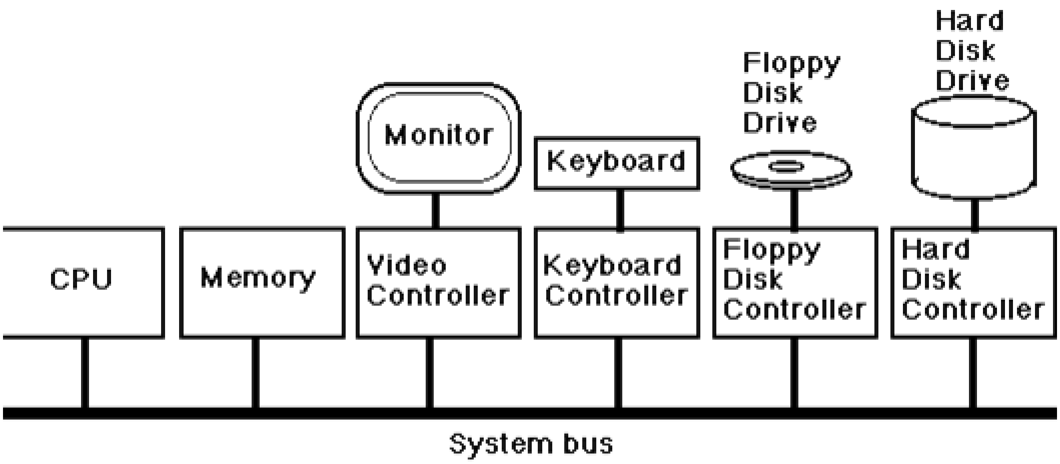
1) Input/output

In computing, **input/output** or **I/O**, is the communication between an information processing system (such as a computer) and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system, and outputs are the signals or data sent from it. The term can also be used as part of an action; to "perform I/O" is to perform an input or output operation. I/O devices are used by a person (or other system) to communicate with a computer. For instance, a keyboard or a mouse may be an input device for a computer, while monitors and printers are considered output devices for a computer. Devices for communication between computers, such as modems and network cards, typically serve for both input and output.



Figure(3): I/O units communication

Variables

Syntax for a variable declaration:

name **DB** *value*

name **DW** *value*

DB – stays for Define Byte.

DW – stays for Define Word.

name – can be any letter or digit combination, though it should start with a letter. It's possible to declare unnamed variables by not specifying the name (this variable will have an address but no name).

value – can be any numeric value in any supported numbering system (hexadecimal, binary, or decimal), or "**?**" symbol for variables that are not initialized.

EX: MOV AL, var1

MOV BX, var2

RET ; stop the program (end)

Var1 DB 7

Var2 DW 1234h

Interrupts

Interrupts can be seen as a number of functions. These functions make the programming much easier, instead of writing a code to print a character you can simply call the interrupt and it will do everything for you. There are also interrupt functions that work with disk drive and other hardware. We call such functions **software interrupts**. Interrupts are also triggered by different hardware, these are called **hardware interrupts**. Currently we are interested in **software interrupts** only.

To make a **software interrupt** there is an **INT** instruction, it has very simple syntax:

INT value

Where **value** can be a number between 0 to 255 (or 0 to 0FFh), generally we will use hexadecimal numbers.

Each interrupt may have sub-functions. To specify a sub-function **AH** register should be set before calling interrupt.

Each interrupt may have up to 256 sub-functions (so we get 256 * 256 = 65536 functions). In general **AH** register is used, but sometimes other registers maybe in use. Generally other registers are used to pass parameters and data to sub-function.

1. **INT 10h**

The following example uses **INT 10h** sub-function **0Eh** to type a "Hello!" message. This functions displays a character on the screen, advancing the cursor and scrolling the screen as necessary.

```
ORG 100h
MOV AH, 0Eh
MOV AL, 'H' ;            ASCII code: 72
INT 10h ; print it!
MOV AL, 'e' ;            ASCII code: 101
INT 10h ; print it!
MOV AL, 'l' ;            ASCII code: 108
INT 10h ; print it!
MOV AL, 'l' ;             ASCII code: 108
INT 10h ; print it!
MOV AL, 'o' ;            ASCII code: 111
INT 10h ; print it!
MOV AL, '!' ;            ASCII code: 33
INT 10h ;                 print it!
RET ;                    returns to operating system.
```

2. INT 21h

INT 21h use many sub-functions **such as:**

- **01h** to read one value of character from keyboard
- **02h** to write one character on the screen.

Every sub-function value was include in AH register

EX1: INT 21h for 01h sub-function

```
MOV AH, 01
INT 21H
```

Ret

In this program, if we read a character key this character was display on the screen, otherwise 0 value was display on screen (such as F1, F4, etc.)

EX2: INT 21h for 02h sub-function

Input: load 02 into AH register

Load ASCII code into DL register

Output: Copy ASCII code into AL register

```
MOV AH , 02H
MOV DL , „?‟
INT 21H
```

………………………………

EX:

```
MOV AH, 1        ; read a character
INT 21H
MOV BL, AL       ; save input character into BL
MOV AH, 2        ; carriage return
MOV DL, 0DH
INT 21H
```

```
MOV DL, 0AH             ; line feed
INT 21H
MOV AH, 2               ; display the character stored in BL
MOV DL, BL
INT 21H
MOV AH, 4CH             ; return control to DOS
INT 21H
```

…………………………………..

EX:

```
MOV ah, 1h ;      keyboard input subprogram
int 21h ;         read character into al
MOV dl, al ;  copy character to dl
MOV ah, 2h ;      character output subprogram
int 21h ;         display character in dl
```

2) CPU:Bus Interface Unit and Execution Unit

The internal function of 8086 processor are partitioned logically into processing units ,Bus Interface Unit(BIU) and Execution Unit (EU).general block diagram of 8086 processor is shown in figure (4).

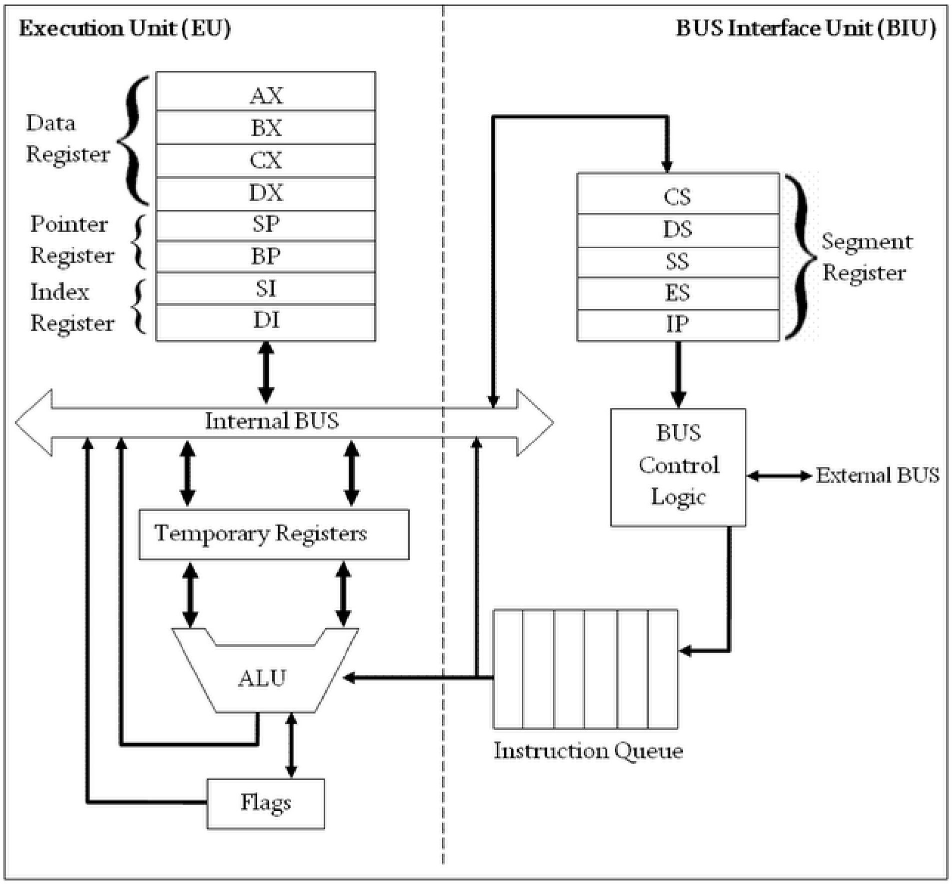


Figure (4): Bus Interface Unit(BIU) and Execution Unit (EU).

Execution Unit (EU) : Execution unit receives program instruction codes and data from the BIU, executes them and stores The results in the general registers. It can also store the data in a memory location or send them to an I/O device by passing the data back to the BIU. This unit, EU, has no connection with the system Buses. It receives and outputs all its data through BIU.

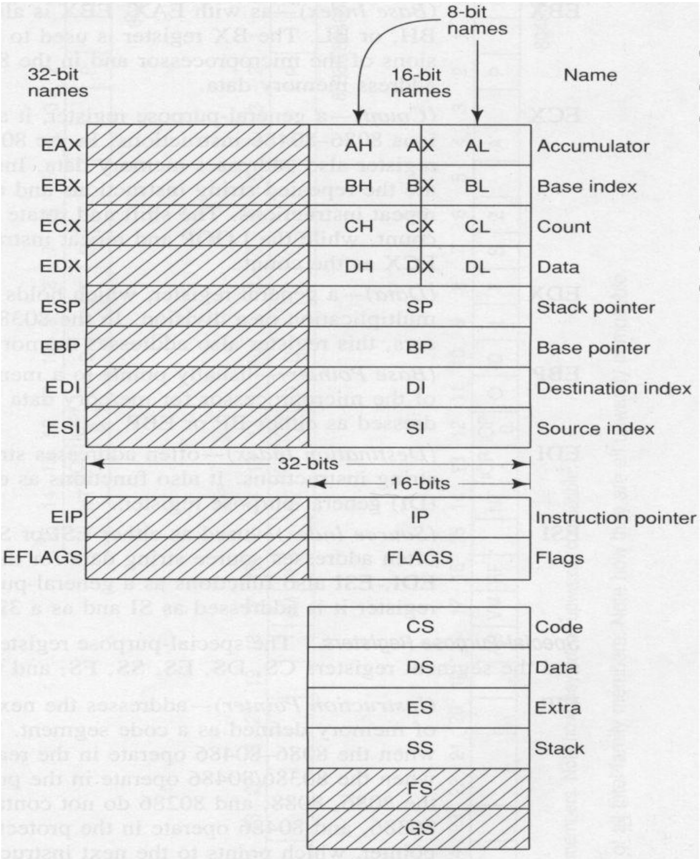
Bus Interface Unit : As the EU has no connection with the system Busses, this job is done by BIU. BIU and EU are connected with an internal bus. BIU connects EU with the memory or I/O circuits. It is responsible for transmitting data, addresses and control signal on the busses.

EU is used mainly to execute instructions. It contains a circuit called the arithmetic and logic unit (ALU). The data for operations are stored in circuit called Registers. The EU has eight registers for storing data; their names are AX, BX, CX, DX, SI, DI, BP, SP and FLAGS register. The EU accepts instructions and data that have been fetched by the BIU and then processes the information. Data processed by the EU can be transmitted to the memory or peripheral devices through the BIU. EU has no direct connection with the outside world and relies solely on the BIU to feed it with instruction and data. It is here that instructions are received, decoded, and executed from the instruction queue portion of BIU. The instructions are taken from the top of the instruction queue on the first-in, first-out, or FIFO, basis.

ALU (Arithmetic & Logic Unit) : This unit can perform various arithmetic and logical operation, if required, based on the instruction to be executed. It can perform arithmetical operations, such as add, subtract, increment, decrement, convert byte/word and compare etc and logical operations, such as AND, OR, exclusive OR, shift/rotate and test etc.

Registers

Registers are 8, 16, or 32-bit high speed storage locations directly inside the CPU, figure(5) designed to be accessed at much higher speed than conventional memory. The CPU has an internal data bus that is generally twice as wide as its external data bus.



Figure(5): Registers 8086 blockdaigram

Figure(6): General Registers of 8086 MPU

- **Index Registers**
    1. **SP (Stack Pointer):** This is stack pointer register pointing to program stack. It is used in conjunction with SS for accessing the stack segment.
    2. **BP (Base Pointer):** This is base pointer register pointing to data in stack segment. Unlike SP, we can use BP to access data in the other segments.
    3. **SI (Source Index):** This is source index register which is used to point to memory locations in the data segment addressed by DS. By incrementing the contents of SI one can easily access consecutive memory locations.
    4. **DI (Destination Index):** This is destination index register performs the same function as SI. There is a class of instructions called string operations.

- **Segment Registers :** BIU has 4 segment busses, CS, DS, SS, ES. These all 4 segment registers holds the addresses of instructions and data in memory. These values are used by the processor to access memory locations. It also contains 1 pointer register IP. IP contains the address of the next instruction to execute by the EU.

1- CS (Code Segment): The code segment register holds the base location of all executable instructions (code) in a program.

2- DS (Data Segment): the data segment register is the default base location for variables. The CPU calculates their location using the segment value in DS.

3- SS (Stack Segment): the stack segment register contain the base location of the stack.

4- ES (Extra Segment): The extra segment register is an additional base location for memory variables.

- **Status and Control register:**

1- IP (Instruction Pointer): The instruction pointer register always contain the offset of the next instruction to be executed within the current code segment. The instruction pointer and the **code segment** register combine to form the complete address of the next instruction.

2- The Flag Register: is a special register with individual bit positions assigned to show the status of the CPU or the result of arithmetic operations. Figure(13) describe the **8086/8088** flags register: