Chapter #1

AN OVERVIEW OF MICROPROCESSORS AND ASSEMBLY LANGUAGE PROGRAMMING.

Chapter One

Learning Outcomes

At the end of the course, student should be:

a) Able to understand the basic operation of microprocessor.

b) Able to understand the basic concept of microprocessor architecture and its pins configuration.

c) Able to understand the machine language programs.

d) Able to design and write programs in assembly language.

e) Able to understand the basic concept of microprocessor input/output interfacing

What is Microprocessor?

- The term ‘micro’ means extremely small and ‘processor’ means the thing that accelerates tasks.
- So in general sense the term ‘microprocessor’ means an extremely small thing that can accelerate different tasks as ordered. But the actual definition of microprocessor is slightly different than ***this***.
- A microprocessor is a tiny electronic chip containing transistors found inside a computer’s central processing unit and other electronic devices. Its basic function is to take input, process it and then provide appropriate output.
- The microprocessor is a general-purpose programmable logic device.
- Understanding the microprocessor concepts is crucial in understanding the operation of digital computer.

The content of the course is divided into three sections:

–microprocessor architecture,

–programming and

–interfacing input/output.

✓The microprocessor itself is usually a single integrated circuit (IC).

✓Most microprocessors (MPU), or very small computers, have much the same commands or instructions that they can perform.

–They vary mostly in the names used to describe each command.

✓In a typical MPU, there are commands to move data around, do simple math (add, subtract, multiply, and divide), bring data into the micro from the outside world, and send data out of the micro to the outside world.

- The microprocessor is a programmable integrated device that has computing and decision-making capability similar to that of the central processing unit (CPU) of a computer.
- The fact that the microprocessor is programmable means it can be instructed to perform given tasks within its capability.
- The microprocessor is a clock-driven semiconductor device consisting of electronic logic circuits manufactured by using either a large-scale integration (LSI) or very-large-scale integration (VLSI) technique.

A typical MPU has three basic parts inside. They are:

–the Program Counter (PC)

–Memory, and

–Input / Output (I/O).

- The Program Counter keeps track of which command is to be executed.
- The Memory contains the commands to be executed.
- The Input / Output handles the transfer of data to and from the outside world (outside the MPU physical package).

There are many other actual parts inside the MPU,

✓Nowadays, the microprocessor is being used in a wide range of products called microprocessor-based products or systems.

✓The microprocessor can he embedded in a larger system, can be a stand alone unit controlling processes, or it can function as the CPU of a computer called a microcomputer.

✓The microprocessor communicates and operates in the binary numbers 0 and 1, called bits.

✓Each microprocessor has a fixed set of instructions in the form of binary patterns called a machine language.

✓It is difficult for humans to communicate in the language of 0 s and 1 s.

✓Therefore, the binary instructions are given abbreviated names, called mnenomics, which form the assembly language for a given microprocessor.

A typical programmable machine can be represented with four components: microprocessor, memory, input, and output.

- These four components work together or interact with each other to perform a given task; thus, they comprise a system.
- The physical components of this system are called hardware.
- A set of instructions written for the microprocessor to perform a task is called a program, and a group of programs is called software.

The microprocessor applications are classified primarily in two categories:

–reprogrammable systems and

–embedded systems.

In reprogrammable systems, such as microcomputers, the microprocessor is used for computing and data processing. These systems include:

–general-purpose microprocessors capable of handling large data, mass storage devices (such as disks and CD-ROMs), and peripherals such as printers; –a personal computer (PC) is a typical illustration.

In embedded systems, the microprocessor is a part of a final product and is not available for reprogramming to the end user. Example:

–copying machine

–washing machine.

–Air-conditioner

–Etc.

Microprocessor, CPU & Microcontroller

❑Microprocessor (MPU) -a semiconductor device (integrated circuit) manufactured by using the LSI technique.

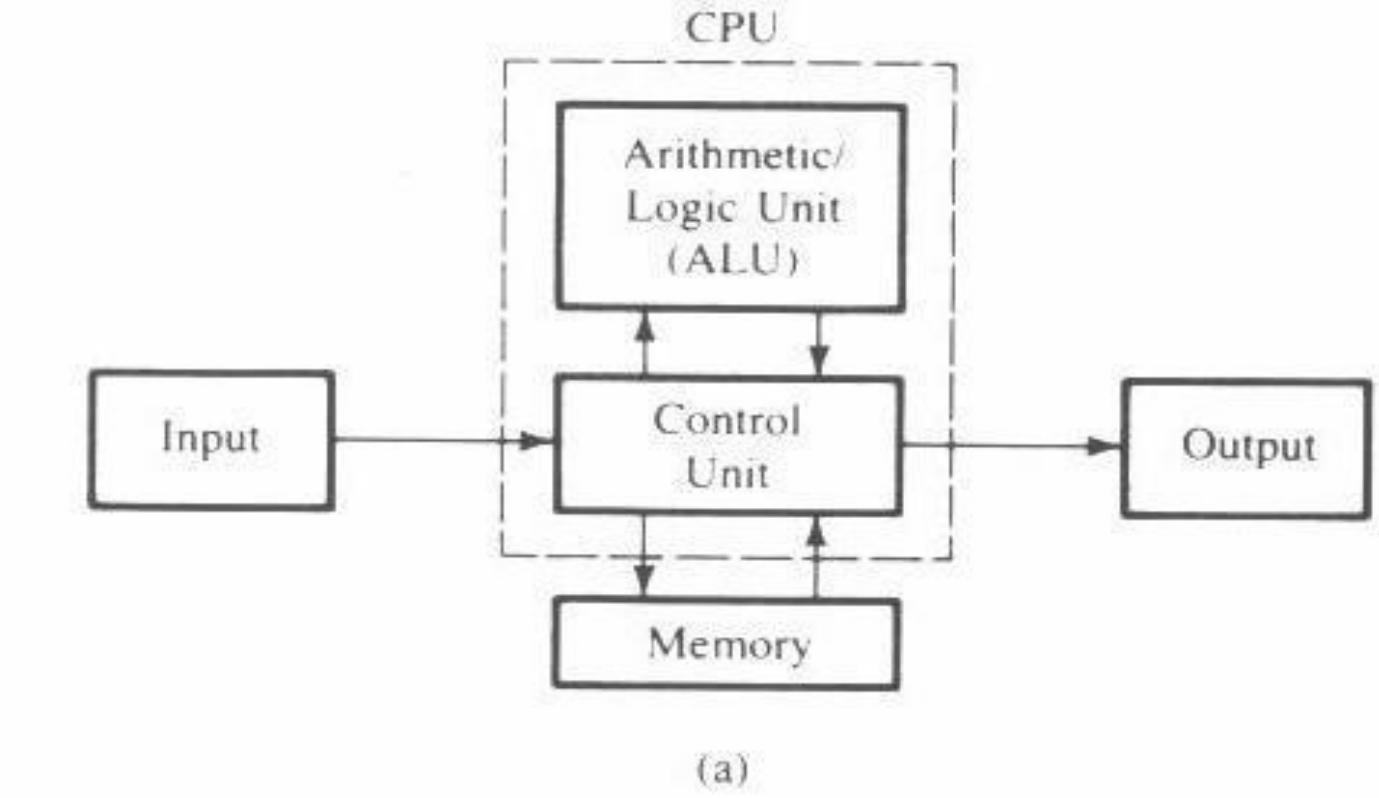
❑It includes the ALU, register arrays, and control circuits on a single chip.

CPU -the central processing unit.

–The group of circuits that processes data and provides control signals and timing. It includes the arithmetic/logic unit, registers, instruction decoder, and the control unit.

❑Microcontroller -a device that includes microprocessor, memory, and I/O signal lines on a single chip, fabricated using VLSI technology.

- In large computers, a CPU implemented on one or more circuit boards performs these computing functions.
- The microprocessor is in many ways similar to the CPU, but includes all the logic circuitry, including the control unit, on one chip.



Traditional block diagram of a computer

Instruction cycle in CPU

I. Read an Instruction

II. Decode the instruction

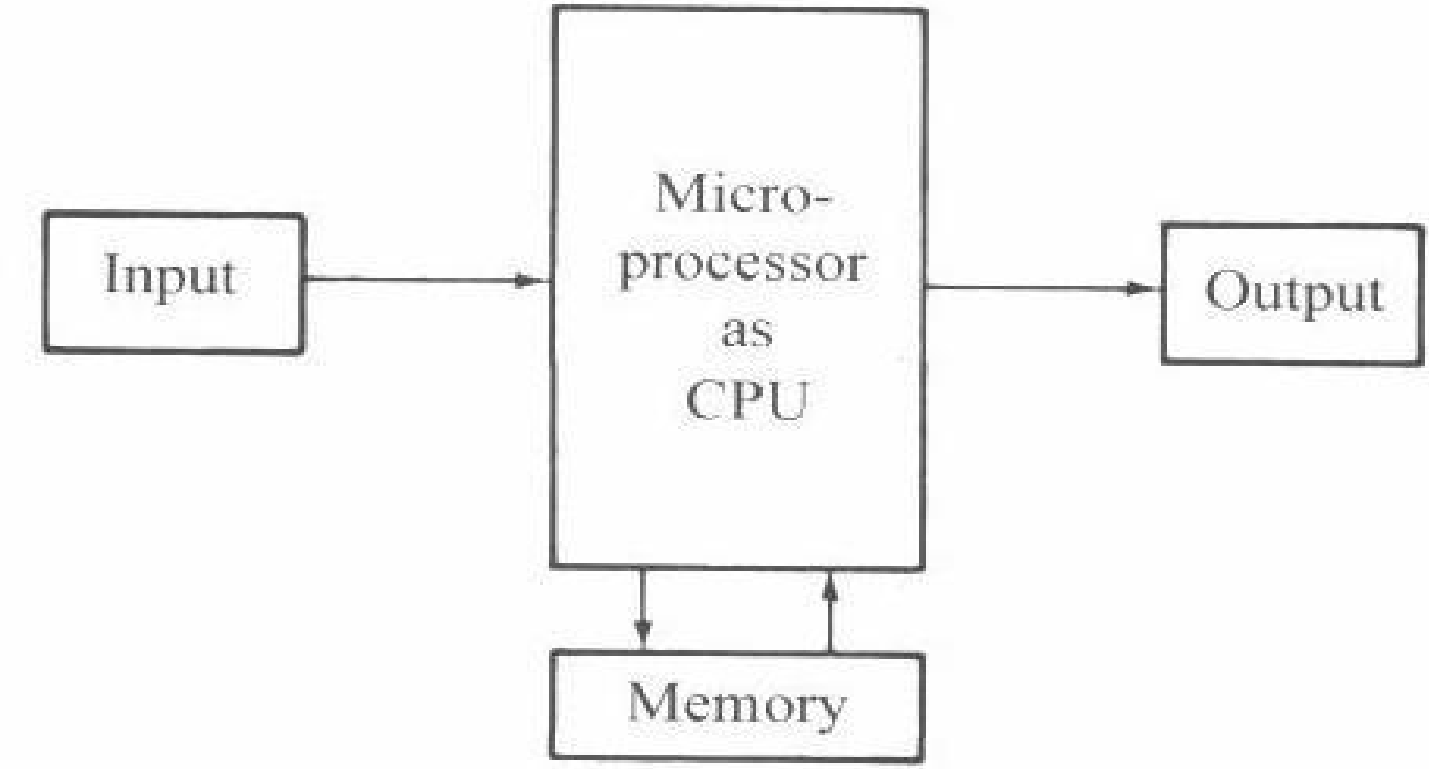
III. Find the address of operand

IV. Retrieve an operand

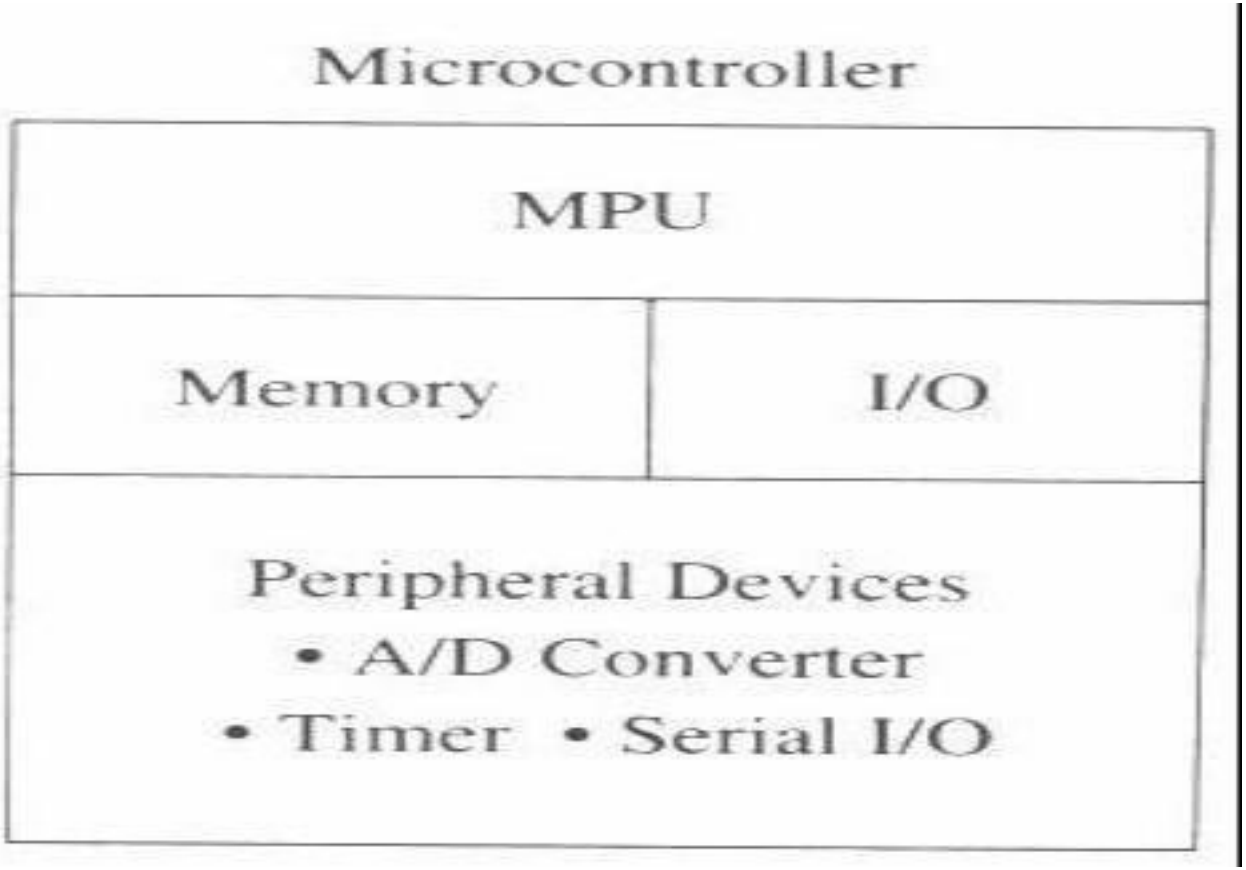
V. Perform desired operation

VI. Find the address of destination

VII. Store the result into the destination.



Block diagram of a computer with the microprocessor as a CPU



Block diagram of a microcontroller

Applications of Microcontrollers

Microcontrollers are widely used in various different devices such as –

i. Light sensing and controlling devices like LED.

ii. Temperature sensing and controlling devices like microwave oven, chimneys.

iii. Fire detection and safety devices like Fire alarm.

iv. Measuring devices like Volt Meter.

Difference between Microprocessor and Microcontroller

Microcontroller	Microprocessor
Microcontrollers are used to execute a single task within an application.	Microprocessors are used for big applications.
Its designing and hardware cost is low.	Its designing and hardware cost is high.
Easy to replace.	Not so easy to replace.
It is built with CMOS technology, which requires less power to operate.	Its power consumption is high because it has to control the entire system.
It consists of CPU, RAM, ROM, I/O ports.	It doesn't consist of RAM, ROM, I/O ports. It uses its pins to interface to peripheral devices.

A Simple Program

- A program is a sequence or series of very simple commands or instructions.
- A real world example program might be the problem of crossing a busy street.

–Step 1: Walk up to the traffic lights and stop.

–Step 2: Look at the traffic light.

–Step 3: Is your light green?

–Step 4: If the light is red, goto step 2. (otherwise continue to step 5)

–Step 5: Look to the left.

–Step 6: Are there cars still passing by?

–Step 7: If yes, goto step 5. (otherwise continue to step 8).

–Step 8: Look to the right.

–Step 9: Are there cars still passing by? (there shouldn't be any by now, but, you never know!)

–Step 10: If yes, goto step 8. (otherwise continue to step 11)

–Step 11: Proceed across the street, carefully!! .

Now this may seem childish at first glance, but this is exactly what you do every time you cross a busy street, that has a traffic light.

- This is also exactly how you would tell a MPU to cross the street, if one could.
- This is what I mean by a sequence or series of very simple steps.
- Taken as a whole, the steps lead you cross a busy intersection, which, if a computer did it, would seem very intelligent.
- It is intelligence, people are intelligent. A programmer that programmed these steps into a MPU, would impart that intelligence to the micro.
- The MPU would not, however, in this case, know what to do when it got to the other side, since we didn't tell it.

In a MPU, the problems are different but the logical steps to solve the problem are similar, that is, a series of very simple steps, leading to the solution of a larger problem.

Also notice that since the steps are numbered, 1 through 11, that is the order in which they're executed.

–The Program Counter (PC), in this case, starting with 1 and ending with 11, doing what each one says.

–The PC automatically advances to the next step, after doing what the current step says, unless a *branch, or jump, is encountered.*

–A branch is an instruction that directs the PC to go to a specific step, **other than the next in the sequence.**

The point of this lesson is to show how a simple set of instructions can solve a bigger problem.

–Taken as a whole, the solution could appear to be more complicated than any of the separate steps it took to solve it.

❖The most difficult problem to be solved in programming a MPU is to define the problem you are trying to solve.

–Sounds silly but I assure you, it's not.

–This is the *Logical Thought Process.*

–It is having a good understanding of the problem you're trying to solve.

Decimal, Binary & Hex

The microprocessor operates in binary digits, 0 and 1, also known as bits.

–Bit is an abbreviation for the term binary digit.

–These digits are represented in terms of electrical voltages in the machine: Generally, 0 represents low voltage level, and 1 represents high voltage level.

❖Each MPU recognizes and processes a group of bits called the word.

–A word is a group of bits the computer recognizes and processes at a time.

❖MPUs are classified according to their word length.

–For example, a processor with an 8-bit word is known as an 8-bit microprocessor, and a processor with a 32-bit word is known as a 32-bit microprocessor.

All numbering systems follow the same rules.

❖Decimal is Base 10, Binary is Base 2, and Hex(adecimal) is Base 16.

❖The base of a system refers to how many possible numbers can be in each digit position.

–In decimal, a single digit number is 0 through 9.

–In binary a single digit number is 0 or 1.

–In hex a single digit number is 0 through 9, A,B,C,D,E, and F.

Advances in Semiconductor Technology

After the invention of the transistor, integrated circuits (ICs) appeared on the scene at the end of the 1950s.

–an entire circuit consisting of several transistors, diodes, and resistors could be designed on a single chip.

- In the early 1960s, logic gates 7400 series were commonly available as ICs, and the technology of integrating the circuits of a logic gate on a single chip became known as small-scale integration (SSI).

As semiconductor technology advanced, more than 100 gates were fabricated on one chip:

–medium-scale integration (MSI).

–Example: a decade counter (7490).

- Within a few years, it was possible to fabricate more than 1000 gates on a single chip

–large-scale integration (LSI).

- Now we are in the era of very-large-scale integration (VLSI) and super-large-scale integration (SLSI).
- The lines of demarcation between these different scales of integration are rather ill defined and arbitrary.

The microprocessor revolution began with a bold and innovative approach in logic design pioneered by Intel engineer Ted Hoff.

❖In 1969, Intel was primarily in the business of designing semiconductor memory.

–it introduced a 64-hit bipolar RAM chip that year.

TABLE 1.1
Intel Microprocessors: Historical Perspective

Processor	Year of Introduction	Number of Transistors	Initial Clock Speed	Address Bus	Data Bus	Addressable Memory
4004	1971	2,300	108 kHz	10-bit	4-bit	640 bytes
8008	1972	3,500	200 kHz	14-bit	8-bit	16 K
8080	1974	6,000	2 MHz	16-bit	8-bit	64 K
8085	1976	6,500	5 MHz	16-bit	8-bit	64 K
8086	1978	29,000	5 MHz	20-bit	16-bit	1 M
8088	1979	29,000	5 MHz	20-bit	8-bit*	1 M
80286	1982	134,000	8 MHz	24-bit	16-bit	16 M
80386	1985	275,000	16 MHz	32-bit	32-bit	4 G
80486	1989	1.2 M	25 MHz	32-bit	32-bit	4 G
Pentium	1993	3.1 M	60 MHz	32-bit	32/64-bit	4 G
Pentium Pro	1995	5.5 M	150 MHz	36-bit	32/64-bit	64 G
Pentium II	1997	8.8 M	233 MHz	36-bit	64-bit	64 G
Pentium III	1999	9.5 M	650 MHz	36-bit	64-bit	64 G
Pentium 4	2000	42 M	1.4 GHz	36-bit	64-bit	64 G

*External 8-bit and internal 16-bit data bus

Organization of a Microprocessor-Based System

It includes three components:

–Microprocessor;

–I/O (input/output) and

–memory (read/write memory and read-only memory).

- These components are organized around a common communication path called a bus.
- The entire group of components is also referred to as a system or a microcomputer system.

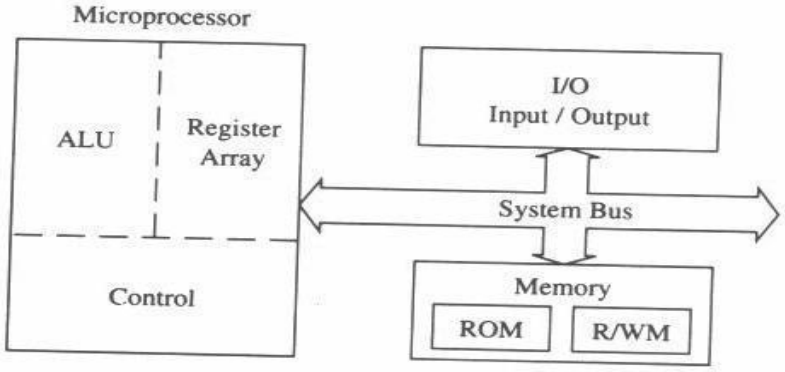


FIGURE 1.3
Microprocessor-Based System with Bus Architecture

The functions of various components:

–The microprocessor

- reads instructions from memory.
- communicates with all peripherals (memory and 1/Os) using the system bus.
- controls the timing of information flow.
- performs the computing tasks specified in a program.

–The memory

- stores binary information, called instructions and data.
- provides the instructions and data to the microprocessor on request.
- stores results and data for the microprocessor.

–The input device

- enters data and instructions under the control of a program such as program.

–The output device

- accepts data from the microprocessor as specified in a program.

–The bus

- carries bits between the microprocessor and memory and 1/Os.

Microprocessor Instruction Set and Computer Languages

- Microprocessors recognize and operate in binary numbers.
  - Each microprocessor has its own binary words, meanings, and language.
  - The words are formed by combining a number of bits for a given machine.
    - The word (or word length) is defined as the number of bits the microprocessor recognizes and processes at a time.
    - The word length ranges from 4-bit to 64-bit.
  - Another term commonly used to express word length is byte.
    - A byte is defined as a group of eight bits.
    - For example, a 16-bit microprocessor has a word length to two bytes.
  - The term nibble stands for a group of four bits.
    - A byte has two nibbles.

Each machine has its own **set of instructions** based on the design of its CPU or of its microprocessor.

- To communicate with the computer, one must give instructions in binary language (machine language).

–Difficult for most people to write programs in sets of 0s and 1s, computer manufacturers have devised English-like words to represent the binary instructions of a machine -assembly language.

–An assembly language is machine-specific.

Assembler

The hand assembly:

–tedious and subject to errors;

–suited for small programs.

❖Alternative, use assembler:

–The assembler is a program that translates the mnemonics entered by the ASCII keyboard into the corresponding binary machine codes of the microprocessor.

–Each microprocessor has its own assembler because the mnemonics and machine codes are specific to the microprocessor being used, and each assembler has rules that must be followed by the programmer.

High-Level Languages

Programming languages that are intended to be machine-independent are called high-level languages.

- These include such languages as BASIC, PASCAL, C, C++ and Java, all of which have certain sets of rules and draw on symbols and conventions from English.
- Instructions written in these languages are known as statements rather than mnemonics.

How are words in English converted into the binary languages of different microprocessors?

–Through another program called either a compiler or an interpreter.

–These programs accept English-like statements as their input, called the source code.

–The compiler or interpreter then translates the source code into the machine language compatible (object code) with the microprocessor being used in the system.

–Each microprocessor needs its own compiler or an interpreter for each high-level language.

❖**Compiler** -a program that translates English-like words of a high-level language into the machine language of a computer.

–A compiler reads a given program, called a source code, in its entirety and then translates the program into the machine language, which is called an object code.

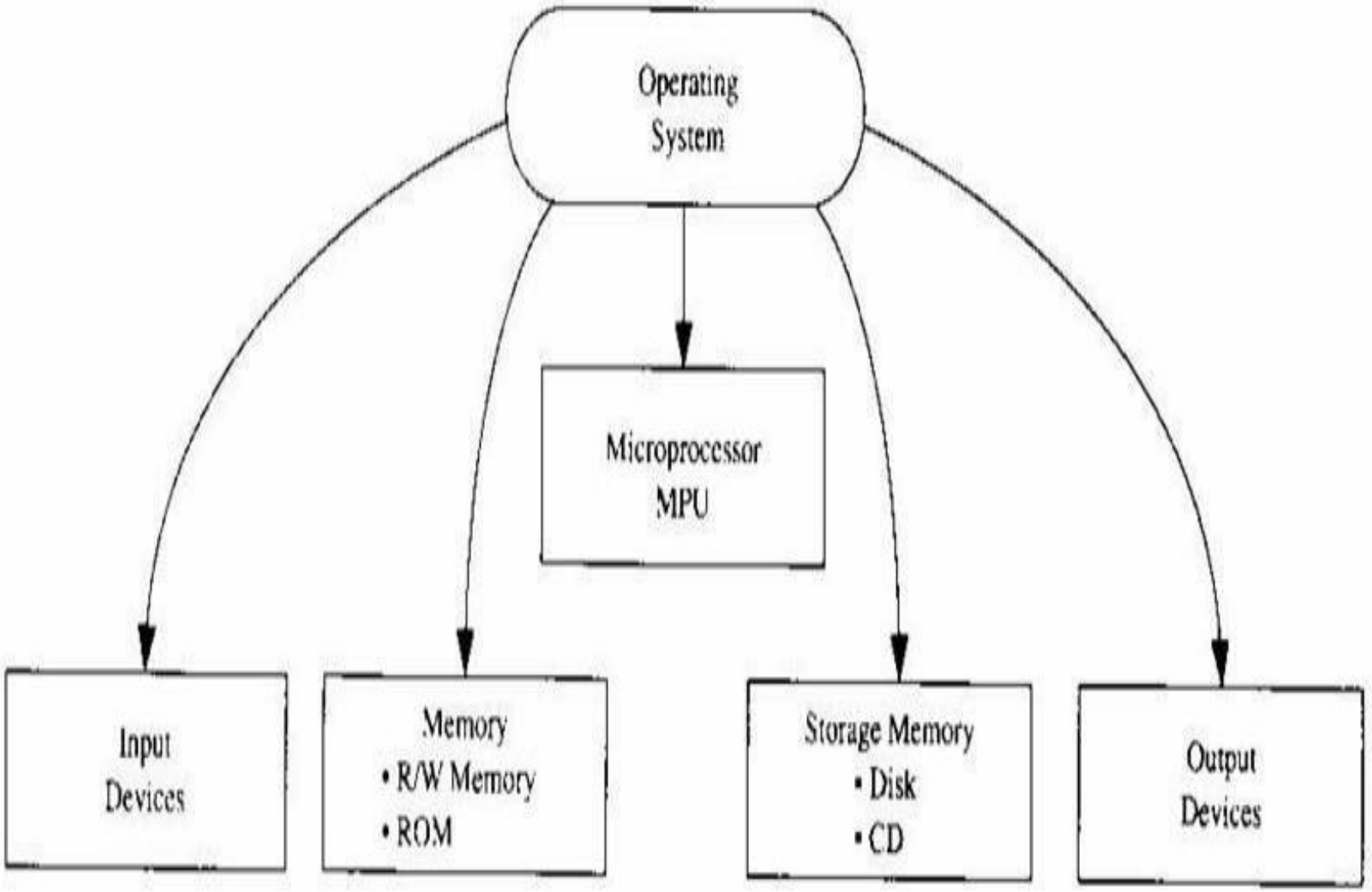
❖**Interpreter** -a program that translates the English-like statements of a high-level language into the machine language of a computer.

–An interpreter translates one statement at a time from a source code to an object code.

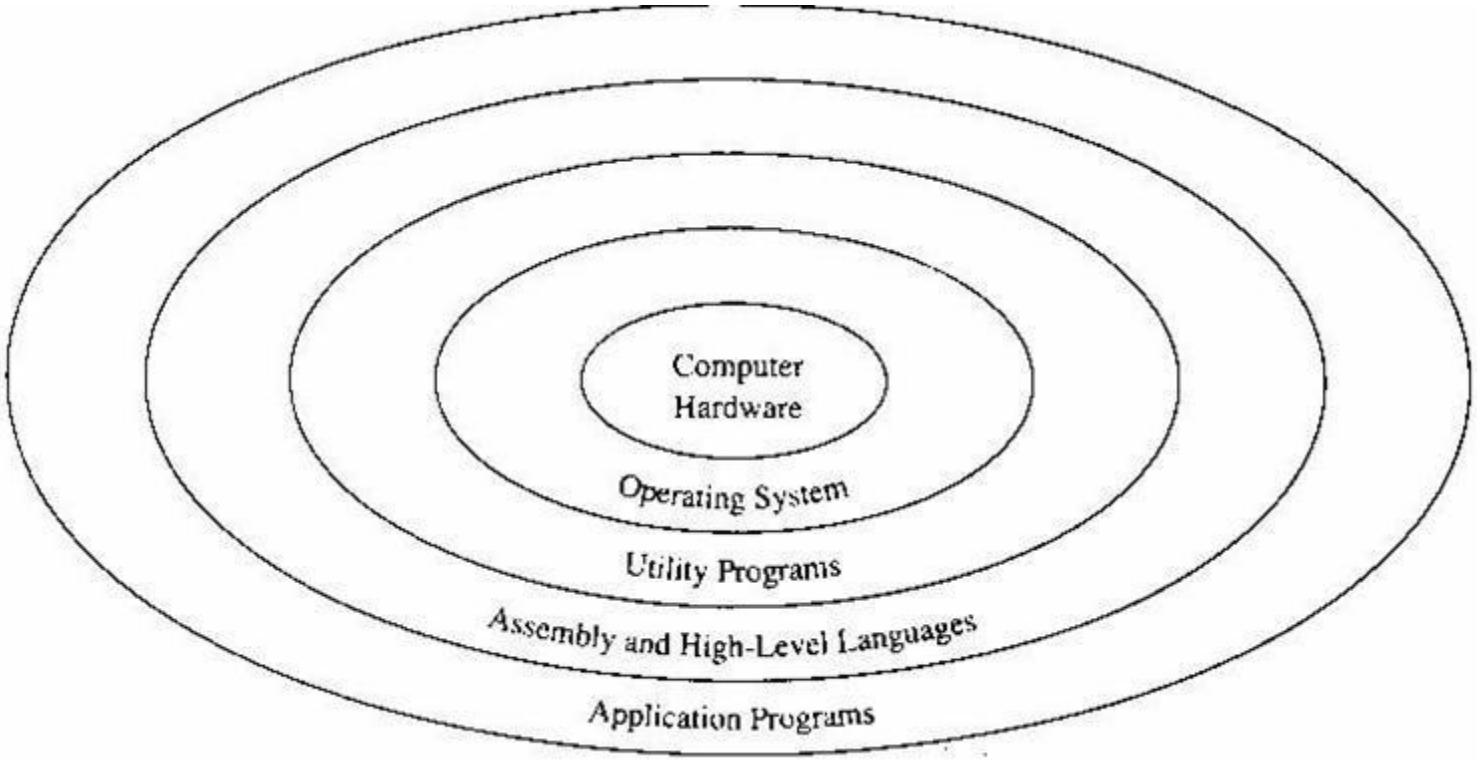
❖**Assembler** -a computer program that translates an assembly language program from mnemonics to the binary machine code of a computer.

Operating system -a set of programs that manages interaction between hardware and software.

–Responsible primarily for storing information on disks and for communication between microprocessor, memory, and peripherals.



OS and its relationship with various hardware components



Hierarchical relationship between computer hardware and software.

Addressing modes of 8086

The way of specifying data to be operated by an instruction is known as **addressing modes**. This specifies that the given data is an immediate data or an address. It also specifies whether the given operand is register or register pair.

Types of addressing modes:

1. Register mode – In this type of addressing mode both the operands are registers.

Example:

MOV AX, BX

XOR AX, DX

ADD AL, BL

2. Immediate mode – In this type of addressing mode the source operand is a 8 bit or 16 bit data. Destination operand can never be immediate data.

Example:

MOV AX, 2000

MOV CL, 0A

ADD AL, 45

AND AX, 0000

Note that to initialize the value of segment register an register is required.

MOV AX, 2000

MOV CS, AX

3. Displacement or direct mode – In this type of addressing mode the effective address is directly given in the instruction as displacement. Example:

MOV AX, [DISP]

MOV AX, [0500]

4. Register indirect mode – In this addressing mode the effective address is in SI, DI or BX.

Example:

MOV AX, [DI]

ADD AL, [BX]

MOV AX, [SI]

5. Based indexed mode – In this the effective address is sum of base register and index register.

Base register: BX, BP

Index register: SI, DI

The physical memory address is calculated according to the base register.

Example:

MOV AL, [BP+SI]

MOV AX, [BX+DI]

6. Indexed mode – In this type of addressing mode the effective address is sum of index register and displacement.

Example:

MOV AX, [SI+2000]

MOV AL, [DI+3000]

7. Based mode – In this the effective address is the sum of base register and displacement.

Example:

MOV AL, [BP+ 0100]

8. Based indexed displacement mode – In this type of addressing mode the effective address is the sum of index register, base register and displacement.

Example:

MOV AL, [SI+BP+2000]

9. String mode – This addressing mode is related to string instructions. In this the value of SI and DI are auto incremented and decremented depending upon the value of directional flag.

Example:

MOVS B MOVS W

10. Input / Output mode – This addressing mode is related with input output operations.

Example: IN A, 45

OUT A, 50

11. Relative mode –

In this the effective address is calculated with reference to instruction pointer.

Example:

JNZ 8 bit address

IP=IP+8 bit address

Introduction to Registers

- A register is a very small amount of very fast memory that is built into the CPU (central processing unit) in order to speed up its operations by providing quick access to commonly used values.
- Registers are the top of the memory hierarchy and are the fastest way for the system to manipulate data.
- Registers are normally measured by the number of bits they can hold, for example, an 8-bit register means it can store 8 bits of data or a 32-bit register means it can store 32 bit of data.
- Registers are used to store data temporarily during the execution of a program. Some of the registers are accessible to the user through instructions. ***Registers are divided in 5 major categories:***

1. General Purpose Registers
2. Pointer Registers
3. Index Registers
4. Segment Registers
5. Flag Registers

General Purpose Registers:

There are four General Purpose Registers named as follows:

1. **AX (Accumulator Register):** commonly used for arithmetic & logic data transfer.

2. **BX (Base Address Register):** used to save the address of memory location.

3. **CX (Count Register):** keeps record of iterations while a LOOP instruction is running.

4. **DX (Data Register):** holds data of the instruction currently being executed.

Pointer Registers:

There are three pointer registers that are used to point toward some memory address.

1. **BP (Base Pointer):** points to the base element of the stack.

2. **SP (Stack Pointer):** always points to the top element of the stack.

3. **IP (Instruction Pointer):** stores the address of the next instruction to

Index Registers:

Index registers are used for indexed addressing and sometimes also used in addition and subtraction. There are two sets of index registers:

1. **SI (Source Index):** used as source index for string operations.

2. **DI (Destination Index):** used as destination index for string operations.

Segment Registers:

Segments are specific areas defined in a program for containing data, code and stack. There are three main segments:

1. **Code Segment:** it contains all the instructions to be executed.

2. **Data Segment:** it contains data, constants and work areas.

3. **Stack Segment:** it contains data and return addresses of procedures or subroutines. It is implemented as a 'stack' data structure.

Flag Registers:

These registers are programmable. It can be used to store and transfer the data from the registers by using instruction.

The common flag bits are:

- **Overflow Flag (OF):** indicates the overflow of a high-order bit (leftmost bit) of data after a signed arithmetic operation.
- **Direction Flag (DF):** determines left or right direction for moving or comparing string data. When the DF value is 0, the string operation takes left-to-right direction and when the value is set to 1, the string operation takes right-to-left direction.
- **Interrupt Flag (IF):** determines whether the external interrupts like keyboard entry, etc., are to be ignored or processed. It disables the external interrupt when the value is 0 and enables interrupts when set to 1.
- **Trap Flag (TF):** allows setting the operation of the processor in single-step mode. The DEBUG program we used sets the trap flag, so we could step through the execution one instruction at a time.

- **Sign Flag (SF):** shows the sign of the result of an arithmetic operation. This flag is set according to the sign of a data item following the arithmetic operation The sign is indicated by the high-order of leftmost bit. A positive result clears the value of SF to 0 and negative result sets it to 1.
- **Zero Flag (ZF):** indicates the result of an arithmetic or comparison operation. A nonzero result clears the zero flag to 0, and a zero result sets it to 1.
- **Auxiliary Carry Flag (AF):** contains the carry from bit 3 to bit 4 following an arithmetic

operation; used for specialized arithmetic. The AF is set when a 1-byte arithmetic

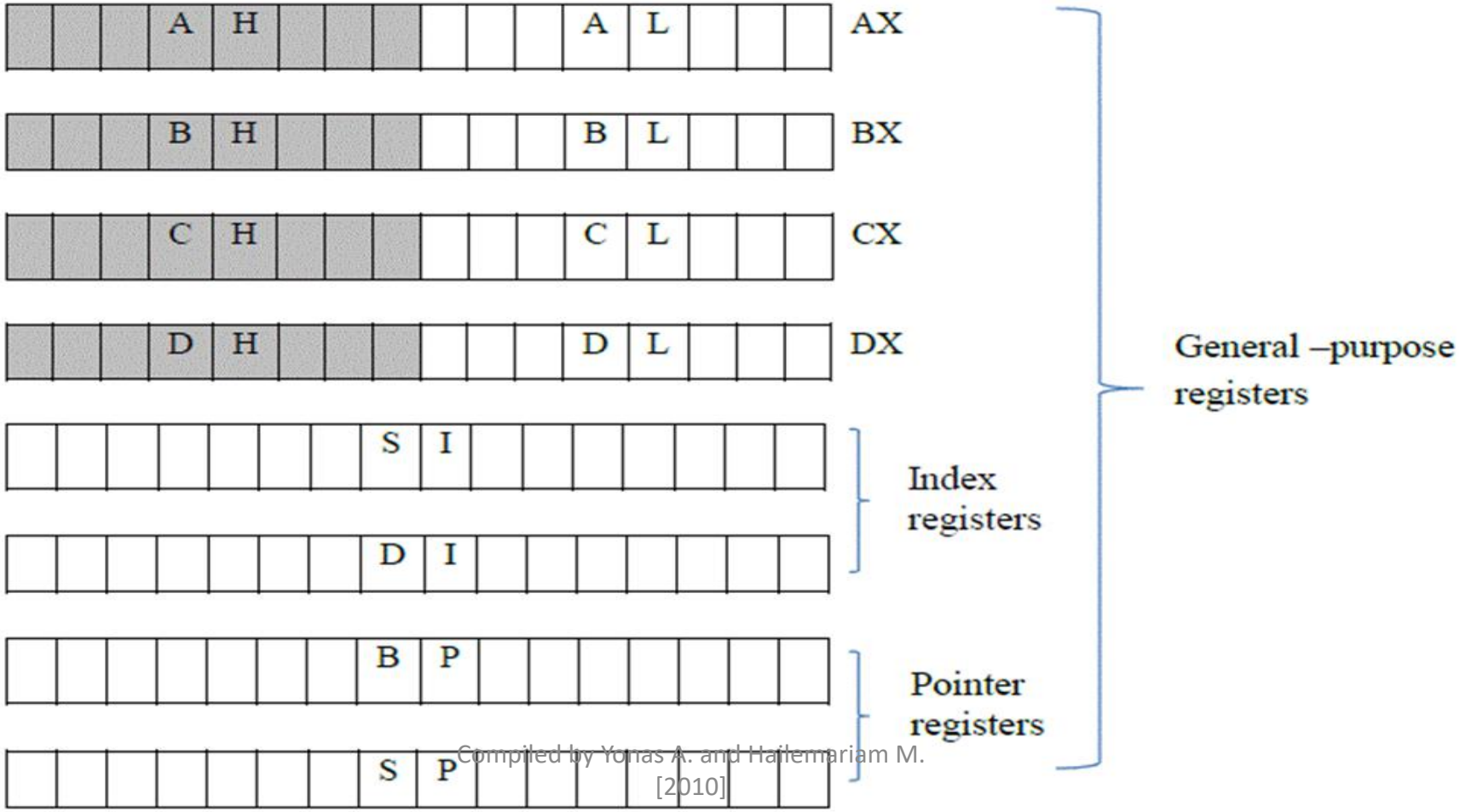
operation causes a carry from bit 3 into bit 4.

- **Parity Flag (PF):** indicates the total number of 1-bits in the result obtained from an arithmetic operation. An even number of 1-bits clears the parity flag to 0 and an odd number of 1-bits sets the parity flag to 1.

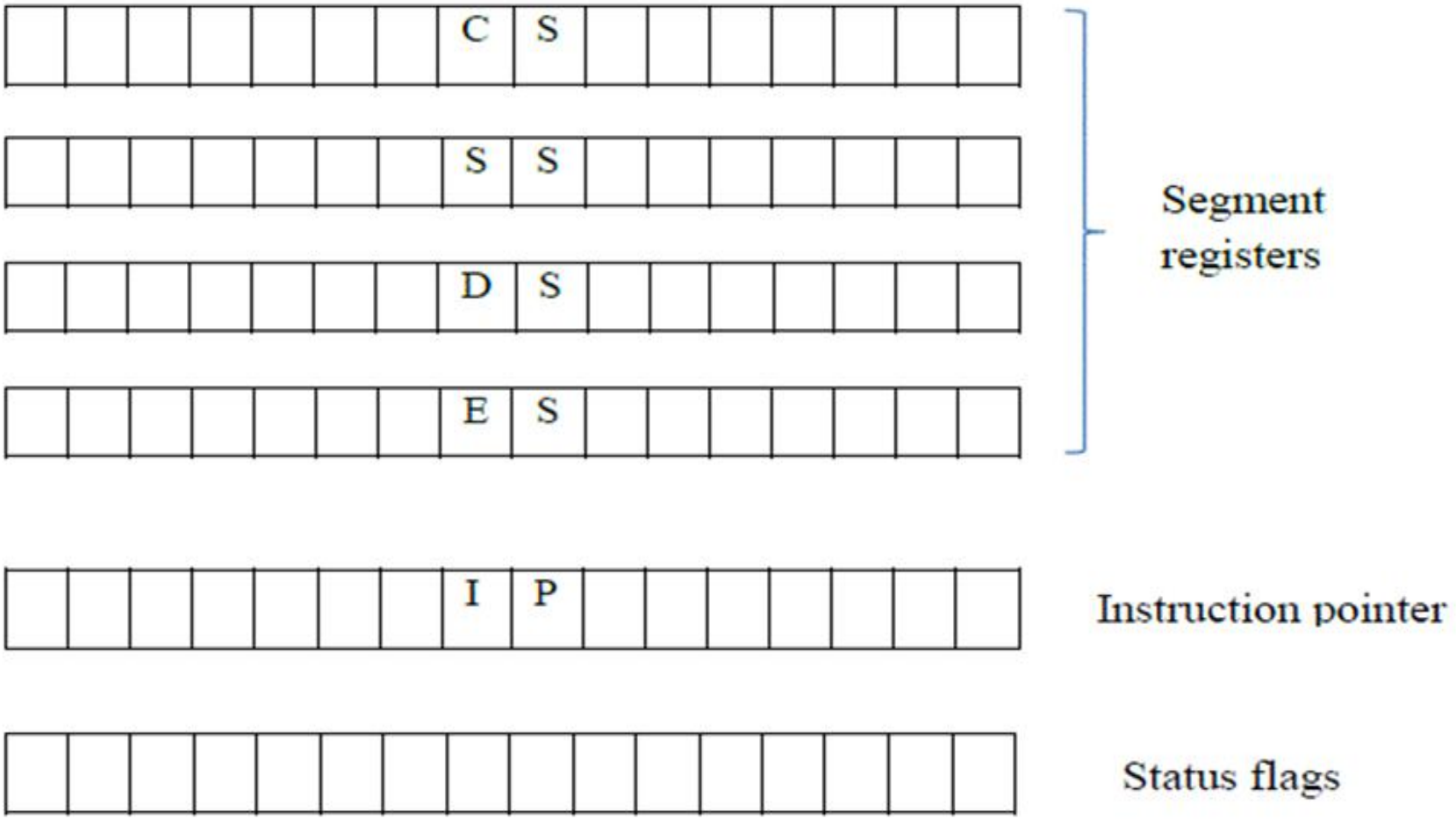
- **Carry Flag (CF):** contains the carry of 0 or 1 from a high-order bit (leftmost) after an arithmetic operation. It also stores the contents of last bit of a shift or rotate operation.

CPU Registers

- The 8086/8088 contains eight general-purpose registers, four segment registers, an instruction



CPU Registers

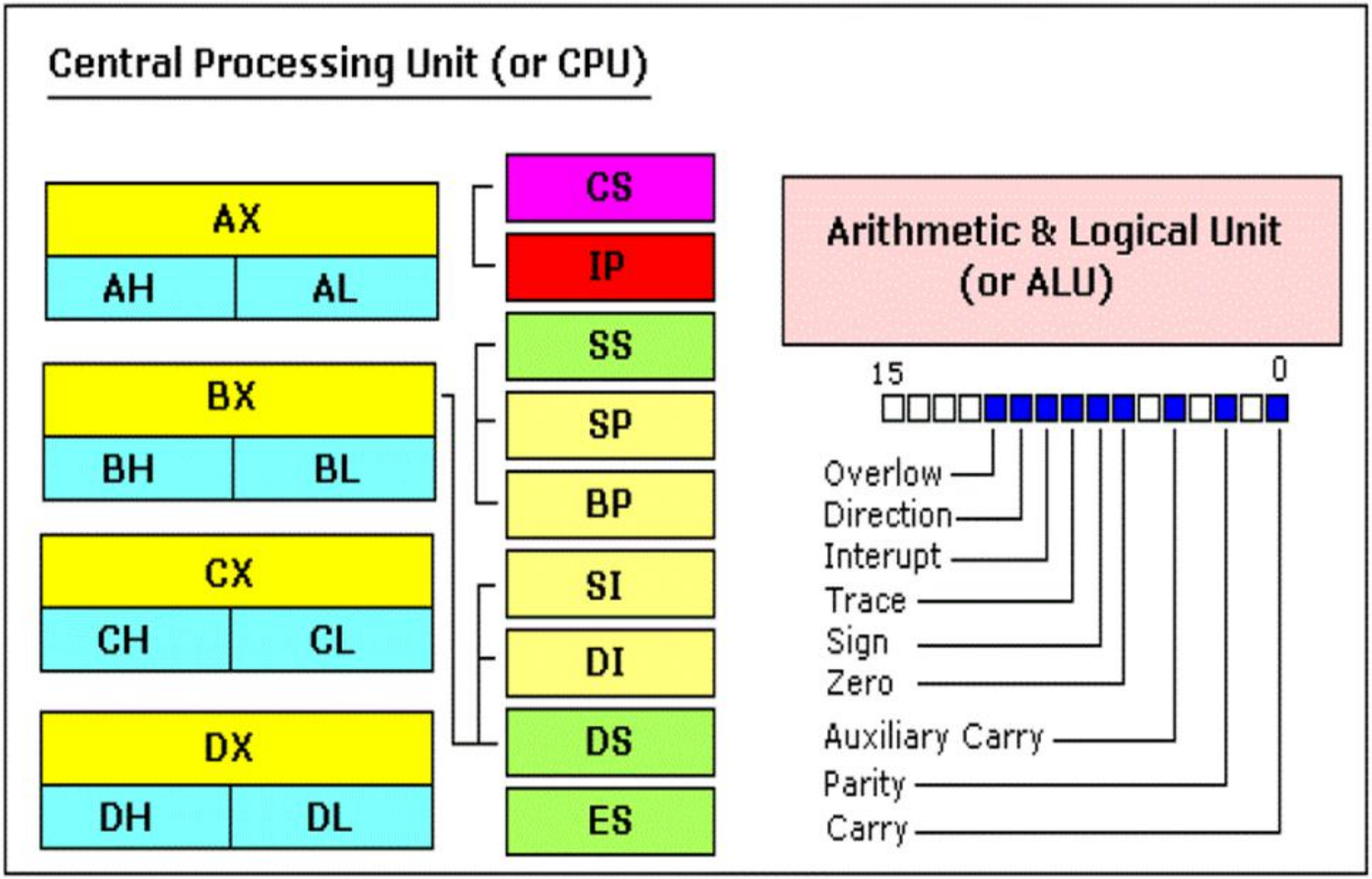


General purpose registers

- 8086 CPU has 8 general purpose registers;
- **AX** - the accumulator register (divided into **AH / AL**).
- **BX** - the base address register (divided into **BH / BL**).
- **CX** - the count register (divided into **CH / CL**).
- **DX** - the data register (divided into **DH / DL**).
- **SI** - source index register
- **DI** - destination index register
- **BP** - base pointer
- **SP** - stack pointer.

- Four general purpose registers **(AX, BX, CX, DX)** are made of two separate 8 bit registers, for example if **AX**= **0011000000111001**, then **AH**=**00110000** and **AL**=**00111001**. Therefore, when you modify any of the 8 bit registers 16 bit register is also updated, and vice-versa. The same is for other three registers, "H" is for high and "L" is for low part.

Inside the CPU



Segment Registers

- **CS (Code Segment)**
  - Points at the segment containing the current program
- **DS (Data Segment)**
  - Generally points at segment where variables are defined.
- **SS (Stack Segment)**
  - Points at the segment containing the stack
- **ES (Extra Segment)**
  - Extra segment register, it's up to a coder to define its usage.

Special purpose registers

- **IP - the instruction pointer**
  - register always works together with **CS** segment register and it points to currently executing instruction.
- **Flags Register** - determines the current state of the processor.
  - is modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program.
  - Generally you cannot access these registers directly

14 types of Registers

- AX - the accumulator register → input/output operations a, ax, eax, rax.
- BX - the base address register→ Holds address of data b, bx, ebx, rbx..
- CX - the count register →counts used in loop cx, ecx, rcx
- DX - the data register →holds data for output d, dx, edx, rdx.
- SI - source index register → points the source operand
- DI - destination index register→ points the destination operand
- BP - base pointer→ base of the top of the stack
- SP - stack pointer → points current top of stack.
- CS (Code Segment)→ holds address of code segment
- DS (Data Segment) → holds address of data segment
- SS (Stack Segment) → holds address of stack segment
- ES (Extra Segment)→ holds address of data segment
- IP (instruction pointer)→holds the next instruction
- Flag registers→ holds current status of the program

X=extended to 16 bits, E=extended to 32 bits, R=rich register to 64 bits

I/O Interfaces

- I/O devices such as keyboards and displays establish communication of computer with outside world.
- Devices can be interfaced in two ways
  - I/O mapped I/O and
  - Memory mapped I/O.
- In I/O mapped I/O, device is identified with a unique device number and data are transferred thru IN/OUT instruction.
- Memory mapped I/O each device is identified with 16 bit address. I/O devices are considered to be a part of memory and memory related instruction is used for data transfer.
- An I/O interface must be able to
  - Determine whether or not it is being interfaced
  - Determine whether it has to send data to CPU or receive data from CPU
  - Send ready signal informing CPU that transfer is over
  - Send interrupt Requests to CPU and receive interrupt acknowledgement and send an interrupt type.

interface

- An Interface can be divided into two parts.
  - A part that interfaces to the I/O device and
  - A part that interfaces to the system bus.
- There must be drivers and receivers to maintain
  - signal quality,
  - logic for translating the interface control signals to proper handshaking signals,
  - logic for decoding address that appear on the bus.
- Handshaking signals are used to determine in which direction transfer has to take place whether from CPU or to CPU.
  - It should determine whether it is a READ or WRITE operation. Interrupt signals also must be handled here.
- Address decoder determine whether it is I/O mapped I/O or Memory mapped I/O from one of the bits. If the decoder finds that an interface is referenced it sends signal to the appropriate device.
- Interfaces can be categorized according to the way I/O devices transfer data either in serial or parallel form.

Memory Mapped

- Memory I/O devices are mapped into the system memory map along with RAM and ROM.
- To access a hardware device, simply read or write to those 'special' addresses using the normal memory access instructions.
- The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device.
- The disadvantage to this method is that the entire address bus must be fully decoded for every device.
  - For example, a machine with a 32-bit address bus would require logic gates to resolve the state of all 32 address lines to properly decode the specific address of any device. This increases the cost of adding hardware to the machine.

I/O Mapped

- I/O devices are mapped into a separate address space. This is usually accomplished by having a different set of signal lines to indicate a memory access versus a port access.
- The advantage to this system is that less logic is needed to decode a discrete address and therefore less cost to add hardware devices to a machine.

Data Movement Instruction

The 8086 microprocessor supports 8 types of instructions –

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Bit Manipulation Instructions
4. String Instructions
5. Program Execution Transfer Instructions (Branch & Loop Instructions)
6. Processor Control Instructions
7. Iteration Control Instructions
8. Interrupt Instructions

1. 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.

2. 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/unsigned double word by word.

IDIV – Used to divide the signed word by byte/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.

3. 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 ≠ 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/MOVSB/MOVSW – Used to move the byte/word from one string to another.

COMS/COMPSB/COMPSW – Used to compare two string bytes/words.

INS/INSB/INSW – Used as an input string/byte/word from the I/O port to the provided memory location.

OUTS/OUTSB/OUTSW – Used as an output string/byte/word from the provided memory location to the I/O port.

SCAS/SCASB/SCASW – Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.

LODS/LODSB/LODSW – 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

4. 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.

5. 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

6. Interrupt Instructions

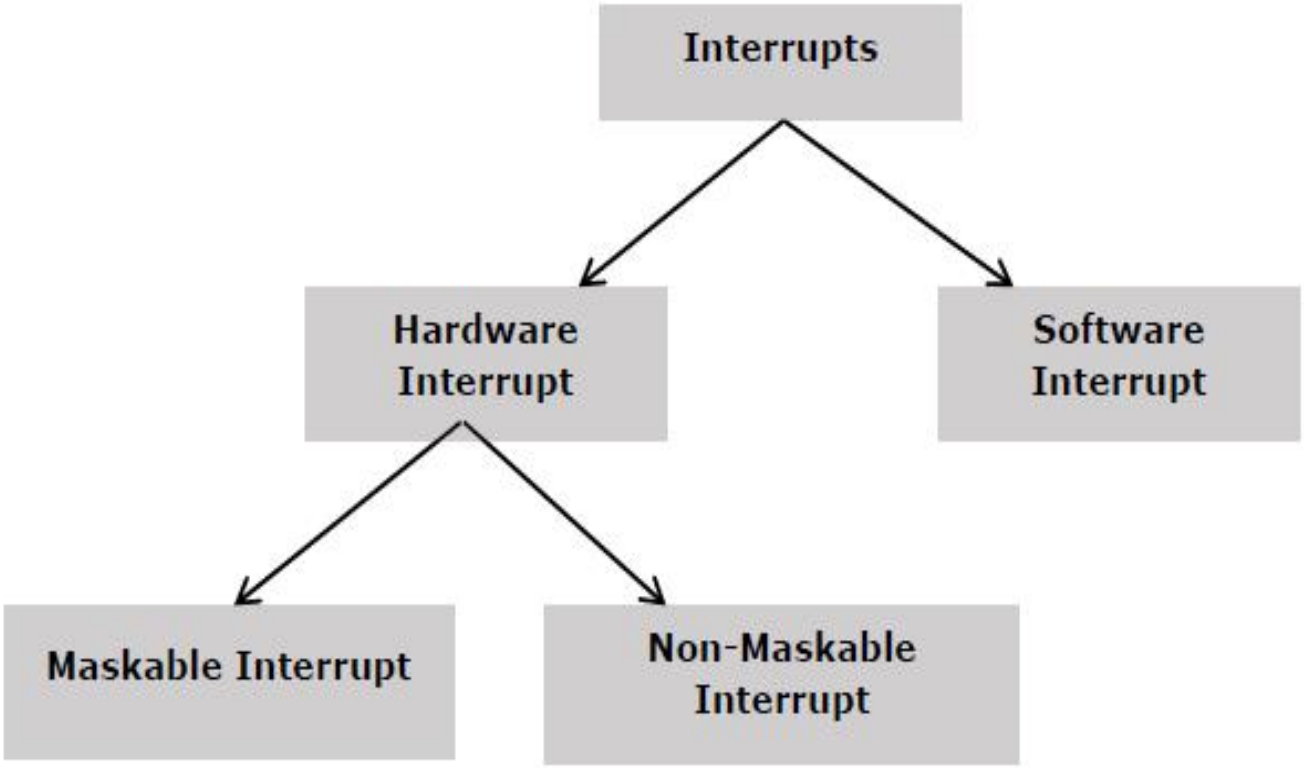
These instructions are used to call the interrupt during program execution.

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

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR**(Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.



Hardware Interrupts

Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR)and it is of type 2 interrupt.

When this interrupt is activated, these actions take place –

i. Completes the current instruction that is in progress

ii. Pushes the Flag register values on to the stack

iii. Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.

iii. IP is loaded from the contents of the word location 00008H

iv. CS is loaded from the contents of the next word location 0000AH.

v. Interrupt flag and trap flag are reset to 0.

INTR

The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction.

The **INTR interrupt** is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends '0' on INTA pin twice.

The first '0' means INTA informs the external device to get ready and during the second '0' the microprocessor receives the 8 bit, say X, from the programmable interrupt controller.

These actions are taken by the microprocessor –

i. First completes the current instruction

ii. Activates INTA output and receives the interrupt type, say X

iii. Flag register value, CS value of the return address and IP value of the return address are pushed on to the stack

iv. IP value is loaded from the contents of word location X × 4

v. CS is loaded from the contents of the next word location

vi. Interrupt flag and trap flag is reset to 0

Software Interrupts

Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers. It includes –

***INT*- Interrupt instruction with type number**

It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. There are 256 interrupt types under this group.

Its execution includes the following steps –

i. Flag register value is pushed on to the stack

ii. CS value of the return address and IP value of the return address are pushed on to the stack

iii. IP is loaded from the contents of the word location 'type number' × 4

iv. CS is loaded from the contents of the next word location

v. Interrupt Flag and Trap Flag are reset to 0

The starting address for type0 interrupt is 000000H, for type1 interrupt is 00004H similarly for type2 is 00008H and .....so on. The first five pointers are **dedicated interrupt pointers**. i.e. –

TYPE 0 interrupt represents division by zero situation.

TYPE 1 interrupt represents single-step execution during the debugging of a program. **TYPE 2** interrupt represents non-maskable NMI interrupt. **TYPE 3** interrupt represents break-point interrupt.

TYPE 4 interrupt represents overflow interrupt.

The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors, and interrupts from 32 to Type 255 are available for hardware and software interrupts.

INT 3-Break Point Interrupt Instruction

It is a 1-byte instruction having op-code is CCH. These instructions are inserted into the program so that when the processor reaches there, then it stops the normal execution of program and follows the break-point procedure.

Its execution includes the following steps –

i. Flag register value is pushed on to the stack

ii. CS value of the return address and IP value of the return address are pushed on to the stack

iii. IP is loaded from the contents of the word location 3×4 = 0000CH

iv. CS is loaded from the contents of the next word location

v. Interrupt Flag and Trap Flag are reset to 0

INTO - Interrupt on overflow instruction

It is a 1-byte instruction and their mnemonic **INTO**. The op-code for this instruction is CEH. As the name suggests it is a conditional interrupt instruction, i.e. it is active only when the overflow flag is set to 1 and branches to the interrupt handler whose interrupt type number is 4. If the overflow flag is reset then, the execution continues to the next instruction.

Its execution includes the following steps –

i. Flag register values are pushed on to the stack.

ii. CS value of the return address and IP value of the return address are pushed on to the stack.

iii. IP is loaded from the contents of word location 4×4 = 00010H

iv. CS is loaded from the contents of the next word location.

v. Interrupt flag and Trap flag are reset to 0

INSTRUCTION FORMATS

- A computer will usually have a variety of instruction code formats.
- It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.
- The most common fields found in instruction formats are:
- **An operation code field:** that specifies the operation to be performed.
  - The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift
- **An address field:** that designates a memory address or a processor registers.
- **A mode field:** that specifies the way the operand or the effective address is determined.
  - The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address.

INSTRUCTION FORMATS

- Computers may have instructions of several different lengths containing varying number of addresses.
- The number of address fields in the instruction format of a computer depends on the internal organization of its registers.
- Most computers fall into one of three types of CPU organizations:
  - Single accumulator organization.
  - General register organization.
  - Stack organization.

Single accumulator organization

- In this type of CPU organization all operations are performed with an implied accumulator register.
- The instruction format in this type of computer uses one address field.
- For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as ADD X.
  - Where X is the address of the operand.
  - The ADD instruction in this case results in the operation
    - AC ← AC + M[X].
  - AC is the accumulator register and M[X] symbolizes the memory word located at address X.

General register organization.

- The instruction format in this type of computer needs three register address fields.
- Thus, the instruction for an arithmetic addition may be written in an assembly language as:
- **ADD R1, R2, R3**
- To denote the operation R1 ← R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers.
- Thus the instruction ADD R1, R2 Would denote the operation R1 ← R1 + R2. Only register addresses for R1 and R2 need be specified in this instruction.
- Computers with multiple processor registers use the move instruction with a mnemonic MOV to symbolize a transfer instruction.

Stack organization.

- Computers with stack organization would have PUSH and POP instructions which require an address field.
- Thus, the instruction PUSH X Will push the word at address X to the top of the stack.
- The stack pointer is updated automatically.
- Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack.
- The instruction ADD In a stack computer consists of an operation code only with no address field.
- This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack.
- There is no need to specify operands with an address field since all operands are implied to be in the stack.

THREE-ADDRESS INSTRUCTIONS

- Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.
- The program in assembly language that evaluates X = (A + B) * (C + D) is shown below.
  - **ADD R1, A, B R1 ← M [A] + M [B]**
  - **ADD R2, C, D R2 ← M [C] + M [D]**
  - **MUL X, R1, R2 M [X] ← R1 * R2**
- It is assumed that the computer has two processor registers, R1 and R2. The symbol M [A] denotes the operand at memory address symbolized by A.
- The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions.
- The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.
- An example of a commercial computer that uses three-address instructions is the Cyber 170. The instruction formats in the Cyber computer are restricted to either three register address fields or two register address fields and one memory address field.

TWO-ADDRESS INSTRUCTIONS

- Two address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word.
- The program to evaluate **X = (A + B) * (C + D)** is as follows:
  - **MOV R1, A R1 ← M [A]**
  - **ADD R1, B R1 ← R1 + M [B]**
  - **MOV R2, C R2 ← M [C]**
  - **ADD R2, D R2 ← R2 + M [D]**
  - **MUL R1, R2 R1 ← R1* R2**
  - **MOV X, R1 M [X] ← R1**
- The MOV instruction moves or transfers the operands to and from memory and processor registers.
- The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

One Address Instructions

- One-address instructions use an implied accumulator (AC) register for all data manipulation.
- For multiplication and division there is a need for a second register. However, here we will neglect the second and assume that the **AC** contains the result of tall operations.
- The program to evaluate X = (A + B) * (C + D) is
  - **LOAD A** AC ← M [A]
  - **ADD B** AC ← A [C] + M [B]
  - **STORE T** M [T] ← AC
  - **LOAD C** AC ← M [C]
  - **ADD D** AC ← AC + M [D]
  - **MUL T** AC ← AC * M [T]
  - **STORE X** M [X] ← AC
- All operations are done between the AC register and a memory operand
- **T** is the address of a temporary memory location required for storing the

Zero Address Instructions

- A stack-organized computer does not use an address field for the instructions ADD and MUL.
- The **PUSH** and **POP** instructions, however, need an address field to specify the operand that communicates with the stack.
- The following program shows how X = (A + B) * (C + D) will be written for a stack organized computer. (TOS stands for top of stack)
  - **PUSH A** **TOS ← A**
  - **PUSH B** **TOS ← B**
  - **ADD** **TOS ← (A + B)**
  - **PUSH C** **TOS ← C**
  - **PUSH D** **TOS ← D**
  - **ADD** **TOS ← (C + D)**
  - **MUL** **TOS ← (C + D) * (A + B)**
  - **POP X** **M [X] ← TOS**
- The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.