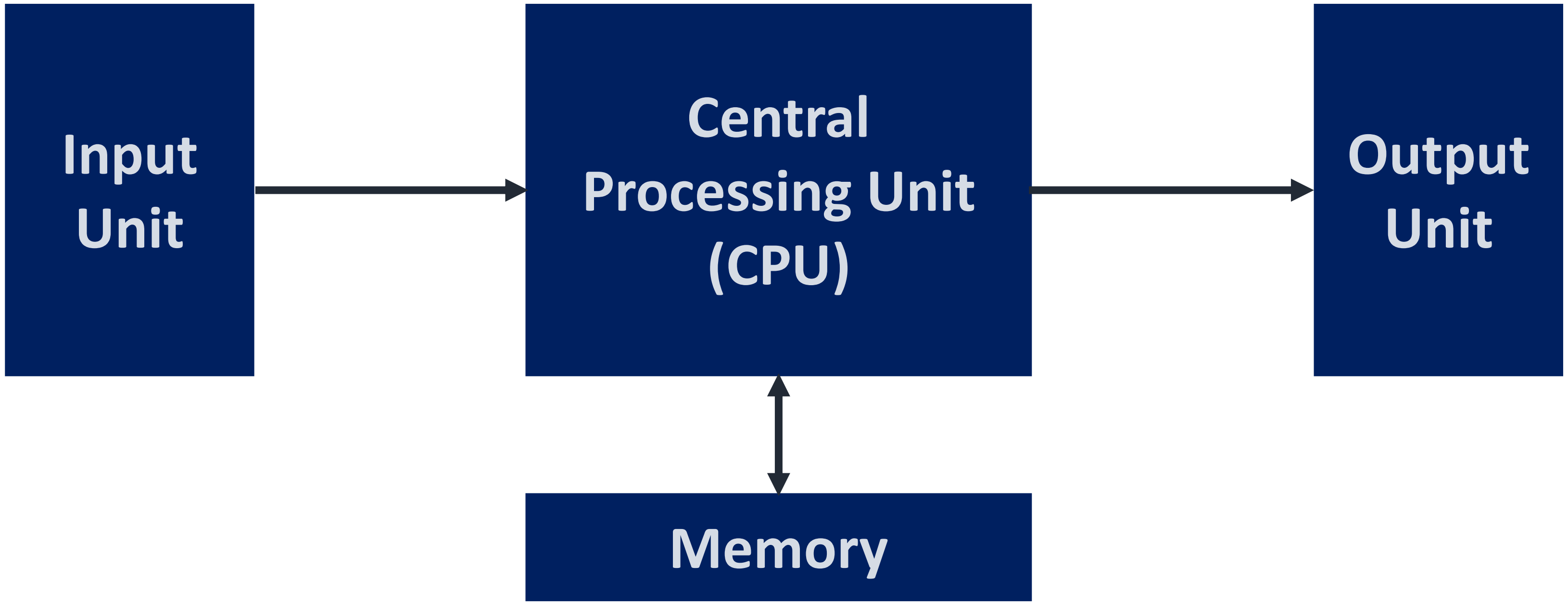
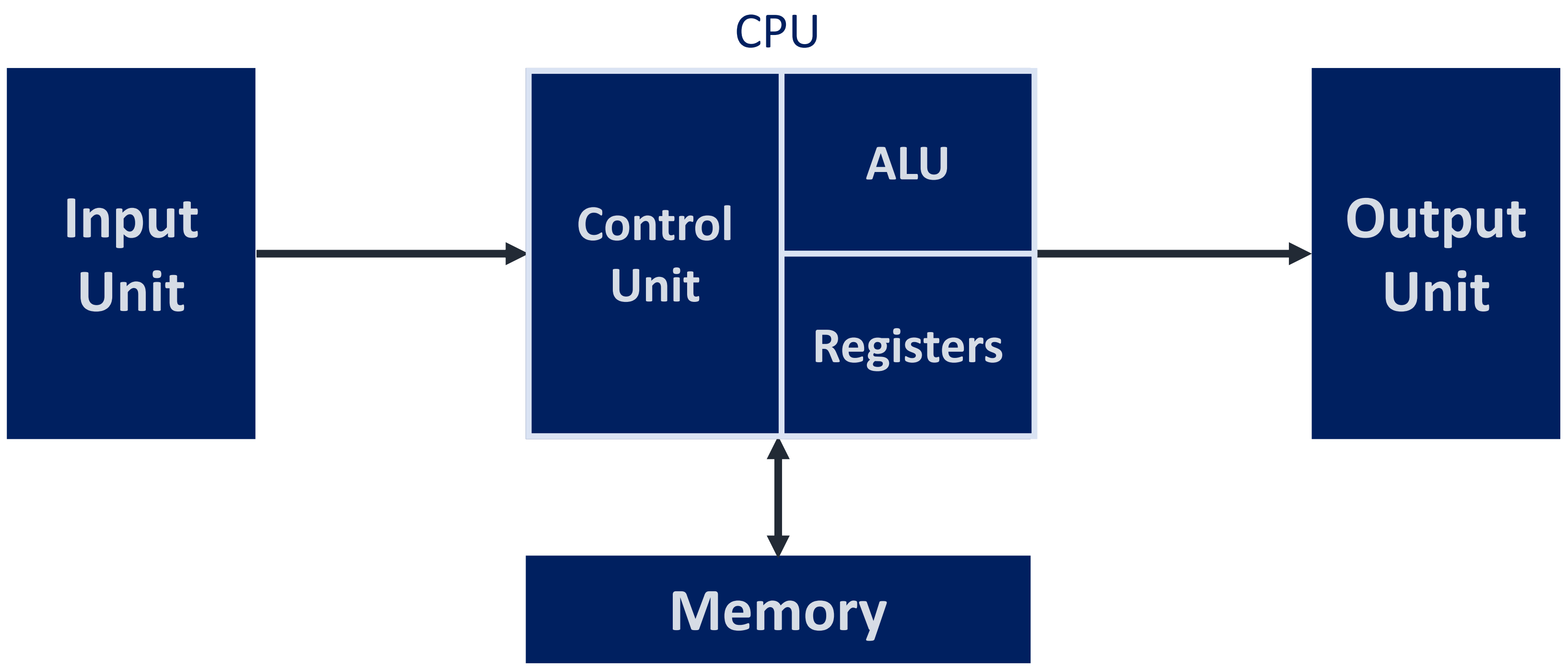
Components of a Microcomputer System

Anika Sayara
anikasayara@outlook.com

Block diagram of a Computer

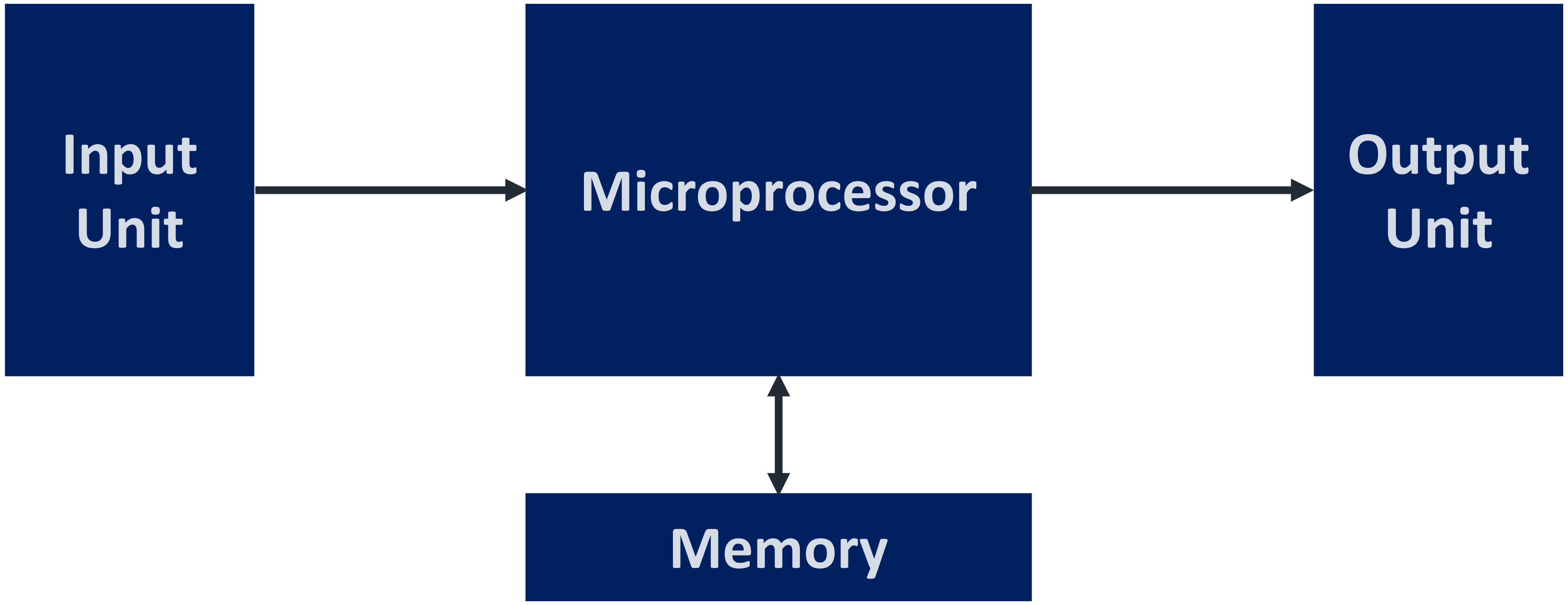


Block diagram of a Computer

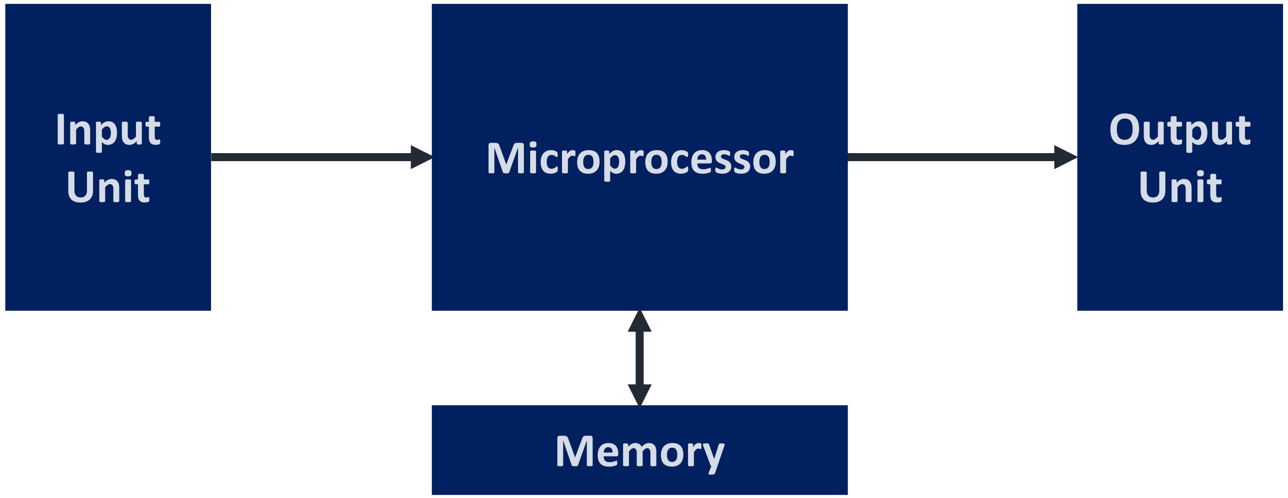


Block diagram of a Microcomputer

The central processing unit of a Microcomputer is contained in a single Integrated Circuit (IC) called the microprocessor



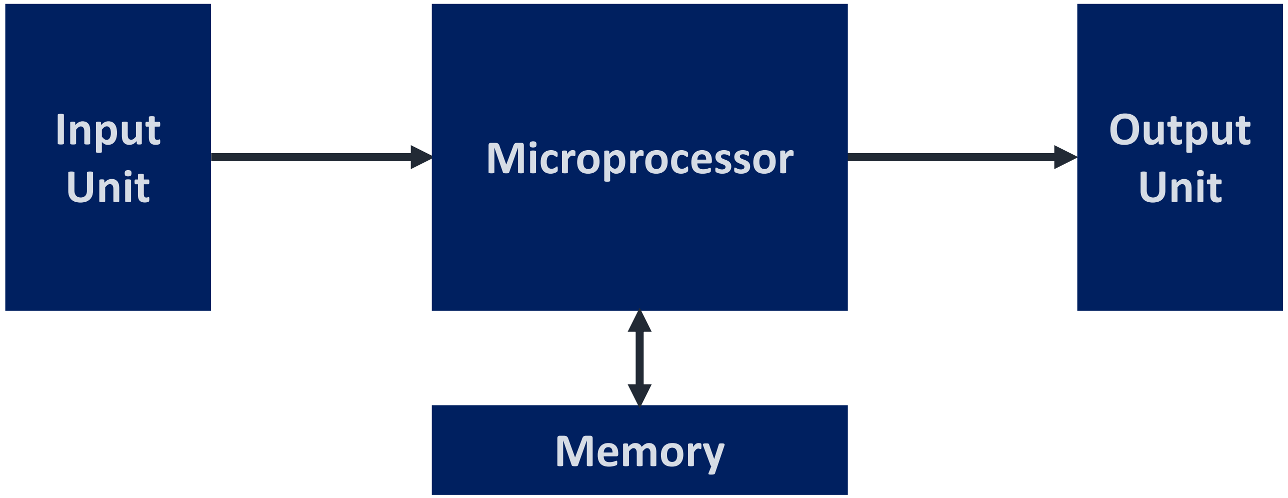
What is a Microcomputer?



A digital computer having microprocessor as its central processing unit along with memory and input and output devices is called a microcomputer.

Examples include Personal Computer, laptop, desktop, tablet, notebook, smartphone, etc.

Components of a typical Microcomputer



- **Memory Unit**
- **Central Processing Unit**
- **Input/Output Units**

Memory

- Information processed by the computer is stored in its memory
- A memory circuit element can store *one bit of data*
- Organized into groups that can store eight bits of data
- String of *eight bits called a byte*
- Each memory byte is *identified by address*
- The first memory byte has address 0
- The data stored in a memory byte called *its contents or values*
- The *address of a memory byte is fixed* and different from any other addresses
- contents are *not unique and subject to change,* because they denote the data *currently* being stored
- The contents of *memory byte are always eight bits but address depends on the processor*. For example some assign 20 bits address whereas some assign 24 bit address

Memory

Address	Contents
.	
.	
.	
.	. . . .
7	0 0 1 0 1 1 0 1
6	1 1 0 0 1 1 1 0
5	0 0 0 0 1 1 0 1
4	1 1 1 0 1 1 0 1
3	0 0 0 0 0 0 0 0
2	1 1 1 1 1 1 1 1
1	0 1 0 1 1 1 1 0
0	0 1 1 0 0 0 0 1

Fig: Organization of memory bytes

Memory

The number of bits used in the address determines the number of bytes that can be accessed by the processor.

Memory

Suppose a processor uses 20 bits for an address. How many memory bytes can be accessed?

Memory

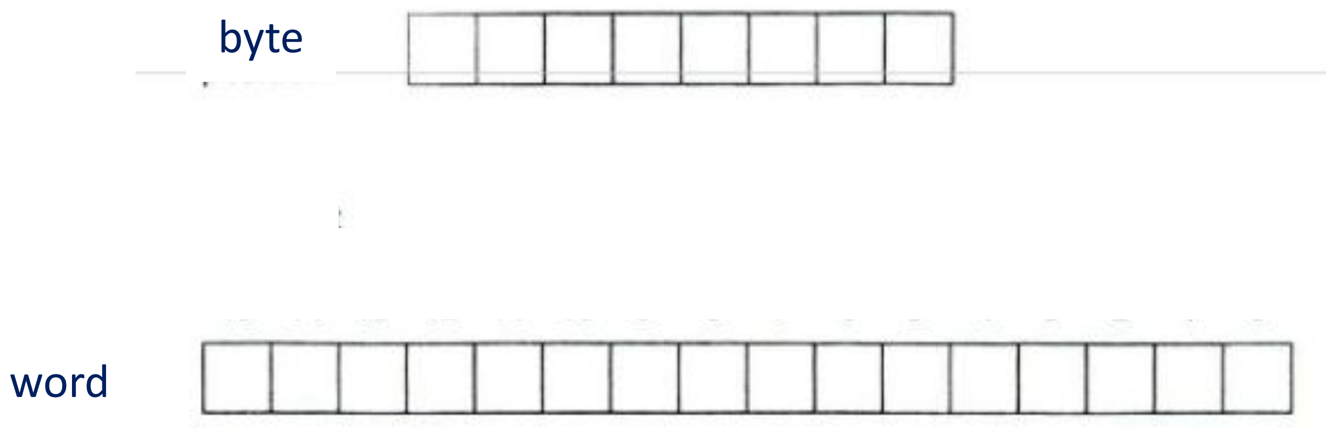
Suppose a processor uses 20 bits for an address. How many memory bytes can be accessed?

A bit can have two possible values – 0 and 1. Hence, 20 bit address can be used to address 2^{20} = 1,048,576 = 1 MB

Memory

Bytes and Words

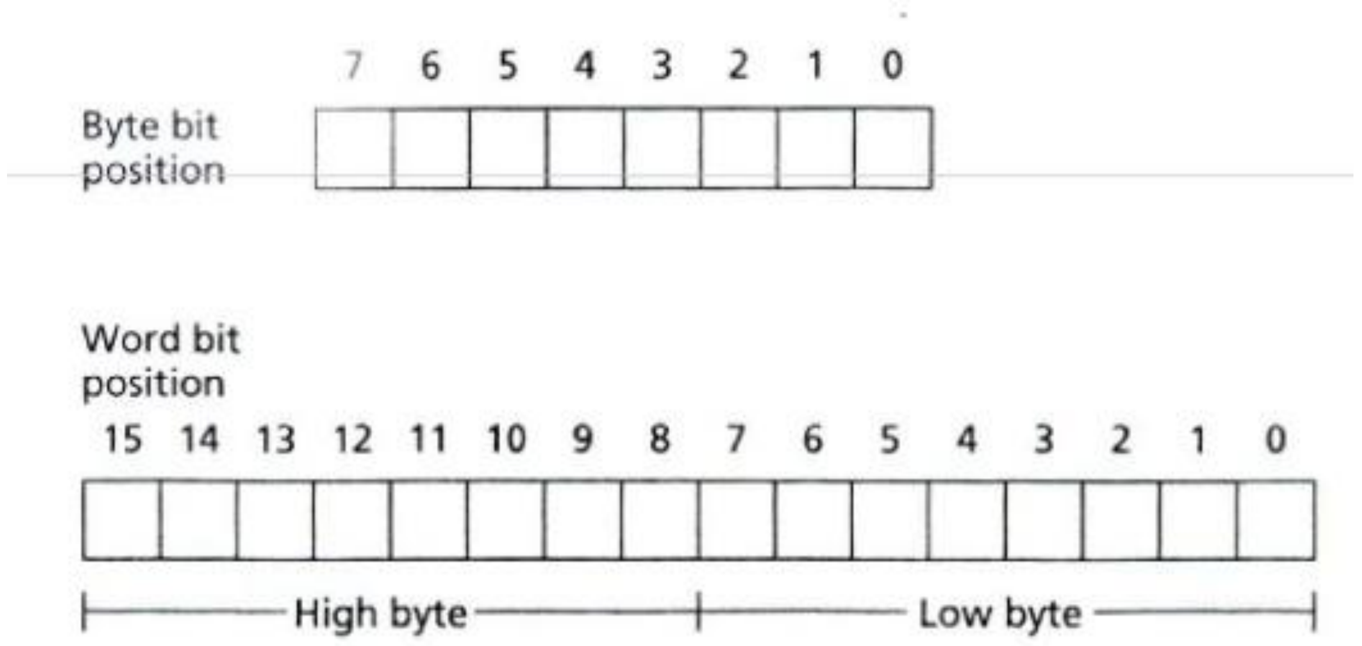
Any pair of successive memory bytes are treated as a single unit called memory word



Memory

Bit Position

- **The positions are numbered from right to left, starting with 0.**
- **In a word, the bits 0 to 7 form the *low byte* and the bits 8 to 15 form the *high byte.***



Memory

RAM and ROM

- Two kinds of memory circuits: *RAM(Random Access Memory)* and *ROM(Read Only Memory)*
- RAM locations can be read and write but ROM locations can only be read
- Program instructions and data being used by the CPU in real time is normally loaded into RAM
- System programs are stored in ROM
- RAM memory is lost when the power is off but ROM circuits retain their values even when the power is off

Memory

Buses

- **Processor *communicates* with memory and I/O devices *by using signals* that travel along a set of wires called buses**
- **Three kinds of signals: address, data and control**
- **Hence, there are three kinds of buses:**
  - ***address bus***
  - ***data bus and***
  - ***control bus***

Memory

Buses

- For example, *to read* the contents of a memory location, the CPU places the *address of the memory location* on the address bus, and it *receives the data*, sent by the memory circuits, on the data bus.
- A control signal is required *to inform the memory* to perform a read operation. The CPU sends the control signal on the control bus.

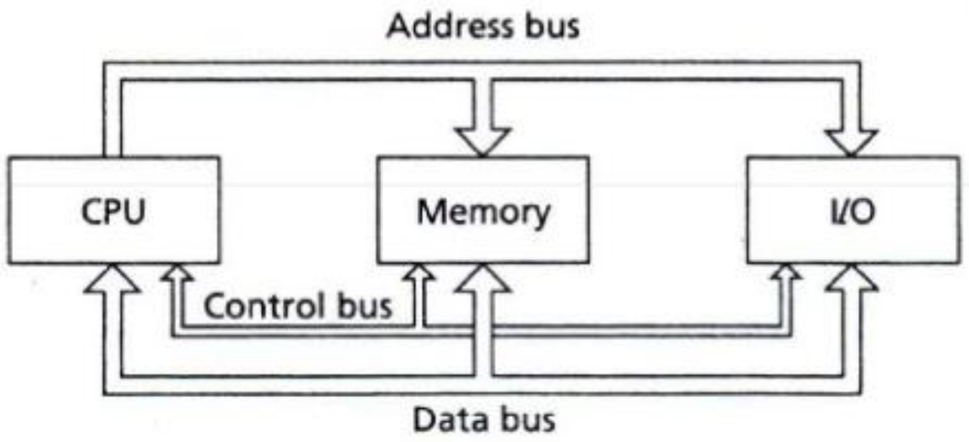


Fig: Bus connections for a microcomputer

The CPU

- **It is the *brain* of the computer**
- **It controls computer by *executing programs* stored in the memory**
- **The instructions performed by a CPU is known as *instruction set***
- **Instruction set for each CPU is *unique***
- **Each instruction that the CPU executes is a *bit string* (for the Intel 8086, instructions are from one to six bytes long).**

The CPU

- **It is the *brain* of the computer**
- **It controls computer by *executing programs* stored in the memory**
- **The instructions performed by a CPU is known as *instruction set***
- **Instruction set for each CPU is *unique***
- **Each instruction that the CPU executes is a *bit string* (for the Intel 8086, instructions are from one to six bytes long).**
- **8086 microprocessor has two main components:**
  - ***Execution Unit (EU) and***
  - ***Bus Interface Unit (BIU)***

The CPU

Execution Unit

- Used to execute instructions
- Contains a circuit called Arithmetic and Logic Unit *(ALU)*
- The ALU can perform arithmetic (+,-,x,/) and logic (AND, OR, NOT) operations
- The data for the operations are stored in circuits called *registers.* (A register is like memory built into the CPU to hold the current data and instructions which are being executed by the CPU).
- Eight registers *for storing data*: AX, BX, CX, DX, SI, DI, BP and SP
- It also contains *temporary registers* for holding operands of the ALU and *FLAGS register* whose individual bits reflect the result of a computation

The CPU

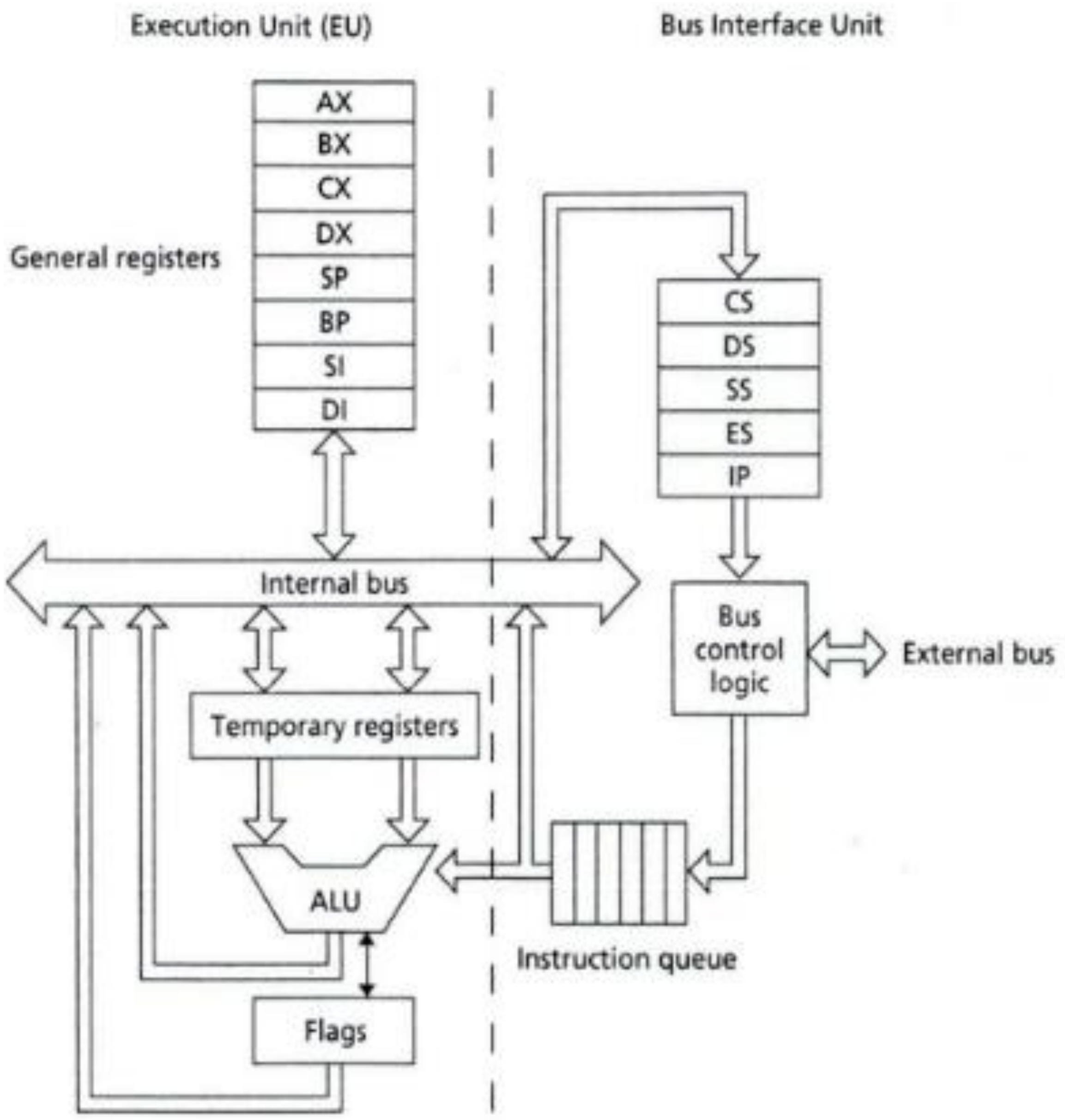
Bus Interface Unit

- ***Facilitates communication*** **between the EU and the memory or I/O circuits**
- ***Transmits*** **addresses, data and control** ***signals*** **on the buses**
- **Registers are CS, DS, ES, SS and IP; they** ***hold addresses*** **of memory locations**
- **The** ***IP (Instruction Pointer)*** **contains the address of the next instruction to be executed by the EU**

The CPU

- **EU and BIU are *connected by an internal bus***
- **When EU is executing an instruction the BIU fetches up to six bytes of the next instruction and places them in the *instruction queue* known as *instruction prefetch***

The CPU


Execution Unit (EU)
Bus Interface Unit
General registers
AX
BX
CX
DX
SP
BP
SI
DI
CS
DS
SS
ES
IP
Internal bus
Bus control logic
External bus
Temporary registers
ALU
Flags
Instruction queue

I/O Ports

- I/O devices *are connected* to the computer *through I/O circuits*
- Each of these circuits contains several registers known as I/O ports.
- I/O ports have addresses and connected to the bus system
- These addresses are known as I/O addresses and can only be used in input or output instructions
- Data to be input from an I/O device are sent to a port where they can be read by the CPU
- On output CPU writes data to an I/O port
- Two types of I/O ports:
  - *Serial and*
  - *Parallel*

I/O Ports

Serial port

- Transfers one bit at a time
- Used for slower transfer such as keyboard

Parallel port

- Transfers 8 or 16 bits at a time
- Requires more wiring connections
- Used for faster data transfer such as disk drives

I/O Devices

Magnetic Disk

Magnetic disks are used for permanent storage of programs and data

- **Floppy Disk**
- **Light weight and portable**
- **Easy to put away for safekeeping and use it on different computers.**
- **Amount of data depends on type, ranging from 360KB-1.44MB (1KB)**
- **Hard Disk**
- **Enclosed in a hermetically sealed container that is non removable from computer called a fixed disk.**
- **Can store more data than floppy disk. Typically 20, 40 to over 100MB.**
- **A program can access information in a hard disk much faster than a floppy disk.**

I/O Devices

Keyboard

- Allows the user to enter information in a computer.
- It has keys of typewriters and a number of control and function keys
- Has own microprocessor that sends coded signal to computer when a key is pressed or released
- No direct contact between keyboard and display

Display Monitor

- Standard output device of the computer
- Displayed information on the screen is generated by video adapter
- Most adapters can generate both text characters and graphics images.
- Some even display in color

I/O Devices

Printers

- Printers are slower than monitors but provide more permanent output
- Printer outputs are known as hardcopies

➢ Daisey wheel

- The output is similar to that of a typewriter

➢ Dot matrix

- Prints characters composed of dots
- Some can generate near-letter-quality printing
- Print characters with different fonts as well as graphics

➢ Laser printers

- Print characters composed of dots
- The resolution is high (300 dots per inch)
- It is expensive

Instruction Execution

Anika Sayara
anikasayara@outlook.com

Machine Instruction

- **A machine instruction has two parts: *Opcode and Operands***
- **Opcode field stands for *operation code* specifies the particular operation that is to be performed. Each operation has its *unique opcode.***
- **Operands fields specify where to get the source and destination operands for the operation specified by the opcode. The source/destination of operands *can be a constant, the memory or one of the general-purpose registers.***

How an instruction is executed?

The CPU goes through the following steps to execute a machine instruction(the fetch-execution cycle):

- **Fetch**
  - **Fetch an instruction from memory**
  - **Decode the instruction to determine the operation**
  - **Fetch data from memory if necessary**

- **Execution**
  - **Perform the operation on the data**
  - **Store the result in memory if needed**

Programming Languages

Anika Sayara
anikasayara@outlook.com

Machine Language

- A CPU can only execute machine language instructions
- Instructions consist of binary code: 1s and 0s

Machine instruction	Operation
10100001 00000000 00000000	Fetch the contents of memory word 0 and put it in register AX.
00000101 00000100 00000000	Add 4 to AX.
10100011 00000000 00000000	Store the contents of AX in memory word 0.

writing programs in machine language is tedious and subject to error!

Assembly Language

- A programming language that uses symbolic names to represent operations, registers and memory locations.
- Readability of instructions is better than machine language
- One-to-one correspondence with machine language instructions to machine code
- Assemblers translates assembly code to machine code

Assembly language instruction	Comment
MOV AX,A	;fetch the contents of location A and put it in register AX
ADD AX, 4	;add 4 to AX
MOV A,AX	;move the contents of AX ;into location A

High Level Language

- **Closer to natural language**
- **Usually machine independant**
- **Compilers translate high-level programs to machine code directly or indirectly via an assembler**

Mapping between high level language and assembly language

- **Translating High Level Language programs to machine language programs is not a one-to-one mapping**
- **A High Level Language instruction (usually called a statement) will be translated to one or more machine language instructions**

Instruction Class	C	Assembly Language
Data Movement	A=5	MOV A,5

Arithmetic or Logic	B=A+5	MOV AX,A
		ADD AX,5
		MOV B,AX

Data Movement	goto LBL	JMP LBL

Advantages of High Level Language

- **high-level languages are *closer to natural languages*, and so it is *easier to read and understand* a high-level language program than an assembly language program.**
- **an assembly language program generally *contains more statements* than an equivalent high-level language program.**
- **Because each computer has its own unique assembly language, assembly language programs are limited to one machine, but a high-level language program *can be executed on any machine that has a compiler for that language.***

Advantages of Assembly Language

- **because assembly language is so close to machine language, a well written assembly language program produces a faster, shorter machine language program.**

- **Assembly Language has the same efficiency of execution as the machine level language. Because this is one-to-one translator between assembly language program and its corresponding machine language program.**

- **directly control the exact instruction sequences the processor executes.**

- **For embedded, real-time applications, one sometimes writes code to run directly on bare iron (i.e., no operating system) - particularly when using single-chip micro-controllers with limited memory. Tightly coded assembler can use less memory**

Why learn assembly language?

- **Accessibility to system hardware**
  - **Assembly Language is useful for implementing system software**
  - **Also useful for small embedded system applications**

- **Space and Time efficiency**
  - **Understanding sources of program inefficiency**
  - **Tuning program performance**
  - **Writing compact code**

- **Writing assembly programs gives the computer designer the needed deep understanding of the instruction set and how to design one**