UNIT I

Computer Evolution: Von Neumann Architecture. Integer Addition and Subtraction ,Floating point representation., Signed numbers, Binary Arithmetic, 1's and 2's Complements , Booths Algorithm, Hardware Implementation, IEEE Standards, Floating Point Arithmetic , The accumulator, Shifts, Carry and Overflow. Instruction Characteristics, CPU with Single BUS, Types of Operands, Types of Operations, Addressing Modes, Instruction Formats.

UNIT II

Processor Organization: Parallelism and Computer arithmetic, Computer arithmetic associatively. Register Organization, 8086 Registers, Instruction Cycles, Addressing Modes. The Instruction cycle, Control of the CPU, Functional Requirements, Single, Two, Three bus structure, Execution of a complete instruction, Branching, Sequencing of Control Signals, Hardwired Control Unit, Micro-Programmed Control.

UNIT III

Memory Organization: Characteristics of Memory Systems, Types of Memory, Design of memory subsystem using Static, Dynamic Memory Chips, Memory interleaving **High Speed Memories:** Cache Memory, Structure of cache and main memory, Elements of Cache Design, Mapping functions, Replacement algorithms, External Memory, Virtual memory

UNIT IV

I/O Organization: Input / Output Module: Need, Techniques, Interrupt Driven I/O, Basic concepts of an Interrupt , Response of CPU to an Interrupt, Design Issues, Priorities, Interrupt handling, Types of Interrupts**.** Data Transfer Techniques, Data Memory Access, Buses, Types of buses, I/O Interface, Synchronous and Asynchronous Data Transfer, Serial I/O, Multiprogramming vs. Multiprocessing

UNIT V

Microprogramming: Basic Principles, Features ,Applications and advantages of microprogramming, Limitations of microprogramming, Parallel Organization, Instruction Set Architecture (ISA), RISC and CISC, Characteristics of CISC, Characteristics of RISC, RISC versus CISC, Vector Processing Requirements and Characteristics of vector processing.

UNIT I

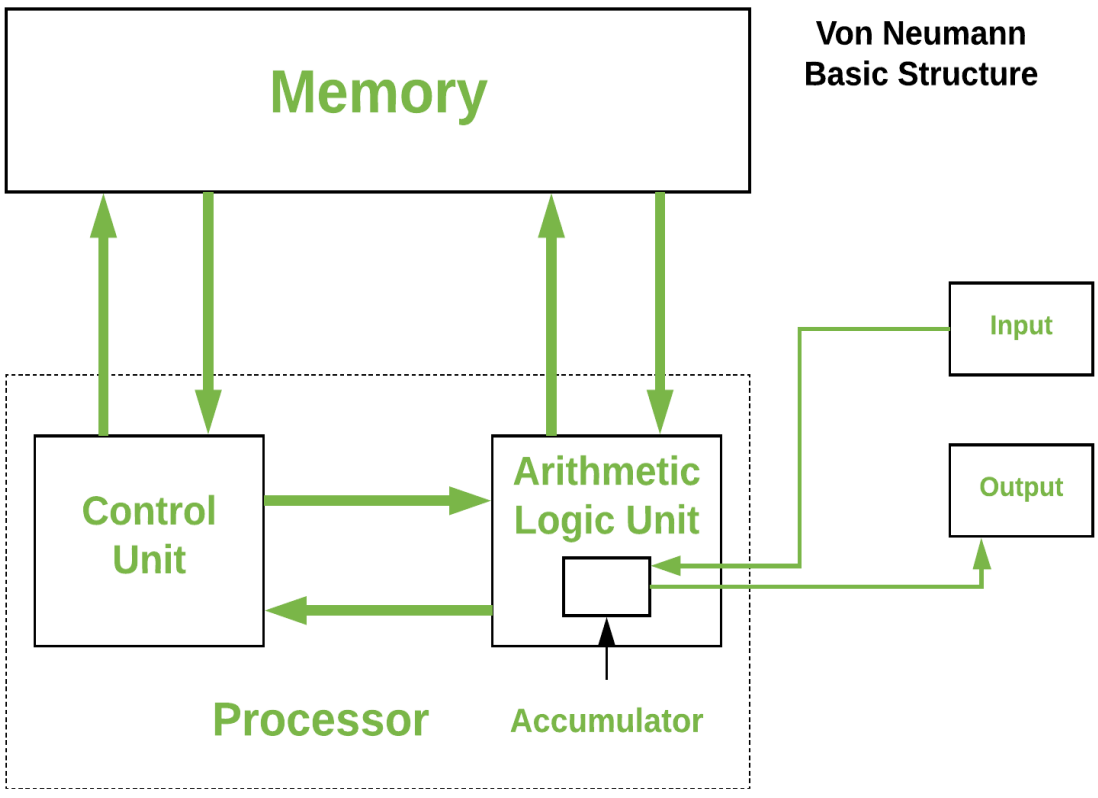
Computer Organization | Von Neumann architecture:

Historically there have been 2 types of Computers:

1. **Fixed Program Computers –** Their function is very specific and they couldn't be programmed, e.g. Calculators.
2. **Stored Program Computers –** These can be programmed to carry out many different tasks, applications are stored on them, hence the name.

The modern computers are based on a stored-program concept introduced by John Von Neumann. In this stored-program concept, programs and data are stored in a separate storage unit called memories and are treated the same. This novel idea meant that a computer built with this architecture would be much easier to reprogram.

The basic structure is like,



It is also known as **IAS** computer and is having three basic units:

The Central Processing Unit (CPU)

1. The Main Memory Unit
2. The Input/Output Device

Let's consider them in details.

- **Control Unit –**
  A control unit (CU) handles all processor control signals. It directs all input and output flow, fetches code for instructions and controlling how data moves around the system.
- **Arithmetic and Logic Unit (ALU) –**
  The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations, and Arithmetic Operation.

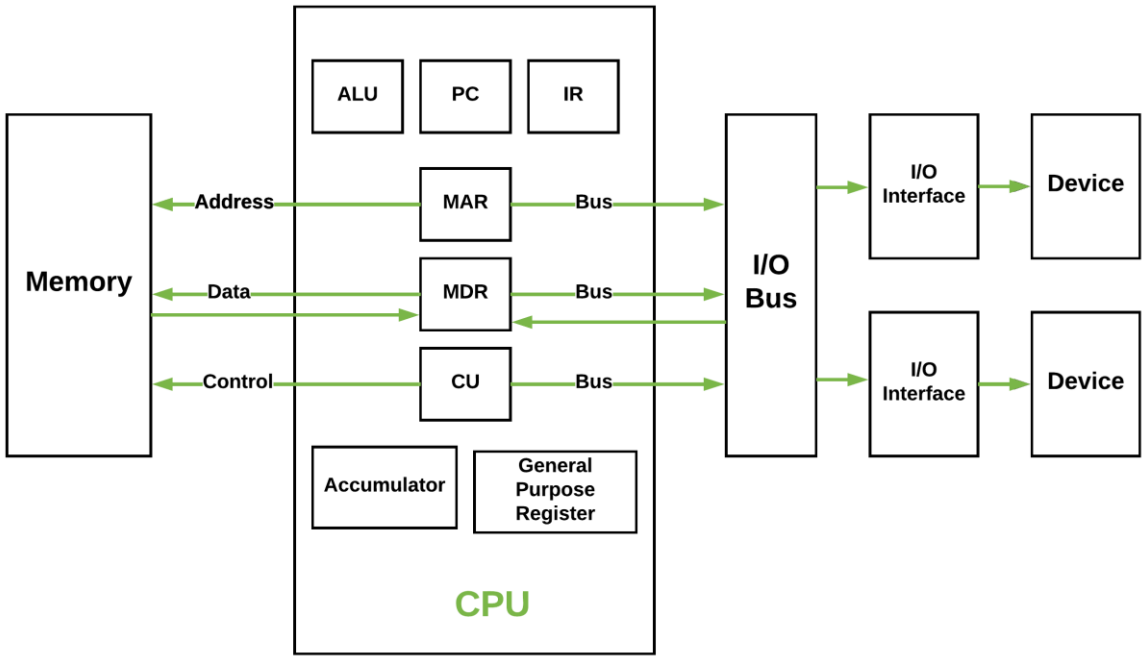


Figure – Basic CPU structure, illustrating ALU

- **Main Memory Unit (Registers) –**

1. **Accumulator:** Stores the results of calculations made by ALU.
2. **Program Counter (PC):** Keeps track of the memory location of the next instructions to be dealt with. The PC then passes this next address to Memory Address Register (MAR).
3. **Memory Address Register (MAR):** It stores the memory locations of instructions that need to be fetched from memory or stored into memory.
4. **Memory Data Register (MDR):** It stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.
5. **Current Instruction Register (CIR):** It stores the most recently fetched instructions while it is waiting to be coded and executed.

6. **Instruction Buffer Register (IBR):** The instruction that is not to be executed immediately is placed in the instruction buffer register IBR.

- **Input/Output Devices –** Program or data is read into main memory from the *input device* or secondary storage under the control of CPU input instruction. *Output devices* are used to output the information from a computer. If some results are evaluated by computer and it is stored in the computer, then with the help of output devices, we can present it to the user.

- **Buses –** Data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory, by the means of Buses. Types:

1. **Data Bus:** It carries data among the memory unit, the I/O devices, and the processor.
2. **Address Bus:** It carries the address of data (not the actual data) between memory and processor.
3. **Control Bus:** It carries control commands from the CPU (and status signals from other devices) in order to control and coordinate all the activities within the computer.

Von Neuman Architecture :

Whatever we do to enhance performance, we cannot get away from the fact that instructions can only be done one at a time and can only be carried out sequentially. Both of these factors hold back the competence of the CPU. This is commonly referred to as the 'Von Neumann bottleneck'. We can provide a Von Neumann processor with more cache, more RAM, or faster components but if original gains are to be made in CPU performance then an influential inspection needs to take place of CPU configuration.

This architecture is very important and is used in our PCs and even in Super Computers.

Integer Addition and Subtraction :

Negative Number Representation

- **Sign Magnitude**

Sign magnitude is a very simple representation of negative numbers. In sign magnitude the first bit is dedicated to represent the sign and hence it is called sign bit.
Sign bit '1' represents negative sign.
Sign bit '0' represents positive sign.

In sign magnitude representation of a n – bit number, the first bit will represent sign and rest n-1 bits represent magnitude of number.

For example,

- +25 = 011001
  Where 11001 = 25
  And 0 for '+'
- -25 = 111001
  Where 11001 = 25

+0 = 000000
– 0 = 100000

2's complement method

To represent a negative number in this form, first we need to take the 1's complement of the number represented in simple positive binary form and then add 1 to it.
For example:
$(8)_{10} = (1000)_2$
1's complement of 1000 = 0111
Adding 1 to it, 0111 + 1 = 1000
So, $(-8)_{10} = (1000)_2$
Please don't get confused with $(8)_{10}$ =1000 and $(-8)_{10}$=1000 as with 4 bits, we can't represent a positive number more than 7. So, 1000 is representing -8 only.

Floating point representation:

Binary Arithmetic, 1's and 2's complement:

1's complement of a binary number is another binary number obtained by toggling all bits in it, i.e., transforming the 0 bit to 1 and the 1 bit to 0.
Examples:

```
Let numbers be stored using 4 bits


1's complement of 7 (0111) is 8 (1000)
1's complement of 12 (1100) is 3 (0011)
```

2's complement of a binary number is 1 added to the 1's complement of the binary number.
Examples:

```
Let numbers be stored using 4 bits


2's complement of 7 (0111) is 9 (1001)
2's complement of 12 (1100) is 4 (0100)
```

These representations are used for signed numbers.

The **main difference** between 1′ s complement and 2′ s complement is that 1′ s complement has two representations of 0 (zero) – 00000000, which is positive zero (+0) and 11111111, which is negative zero (-0); whereas in 2′ s complement, there is only one representation for zero – 00000000 (+0) because if we add 1 to 11111111 (-1), we get 00000000 (+0) which is the same as positive zero. This is the reason why 2′ s complement is generally used.

Another difference is that while adding numbers using 1′ s complement, we first do binary addition, then add in an end-around carry value. But, 2′ s complement has only one value for zero, and doesn't require carry values.

Booths Algorithm, Hardware Implementation, IEEE Standards,

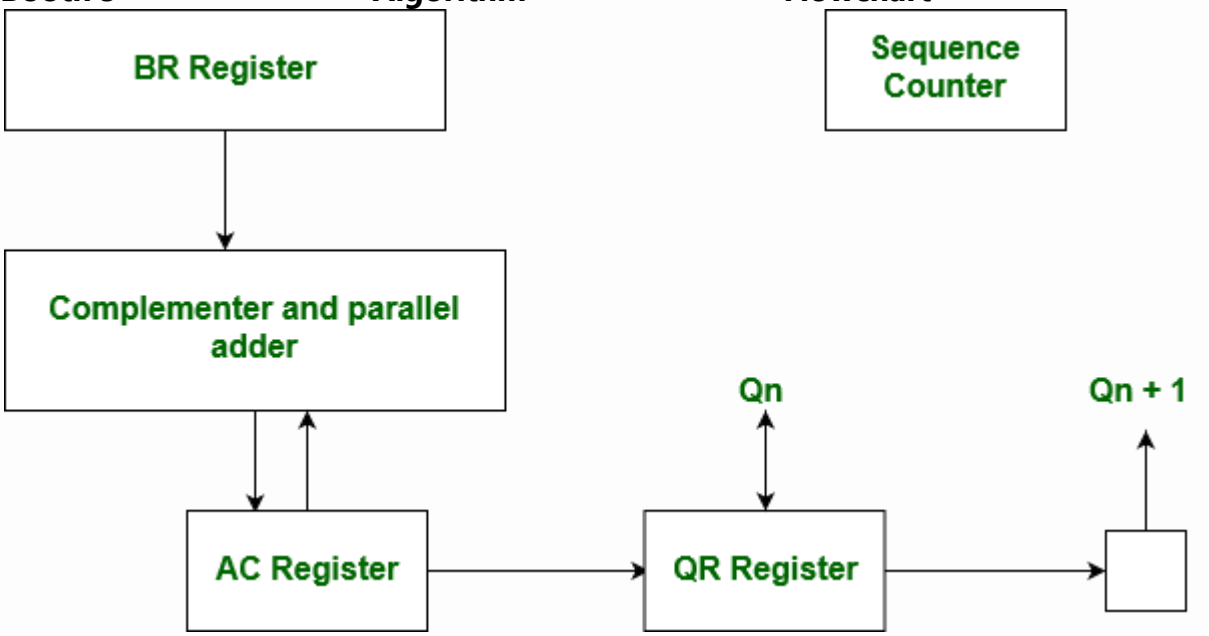
Booth algorithm gives a procedure for **multiplying binary integers** in signed 2's complement representation **in efficient way**, i.e., less number of additions/subtractions required. It operates on the fact that strings of 0's in the multiplier require no addition but just shifting and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as 2^(k+1) to 2^m.

As in all multiplication schemes, booth algorithm requires examination **of the multiplier bits** and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier
2. The multiplier is added to the partial product upon encountering the first 0 (provided that there was a previous '1') in a string of 0's in the multiplier.
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

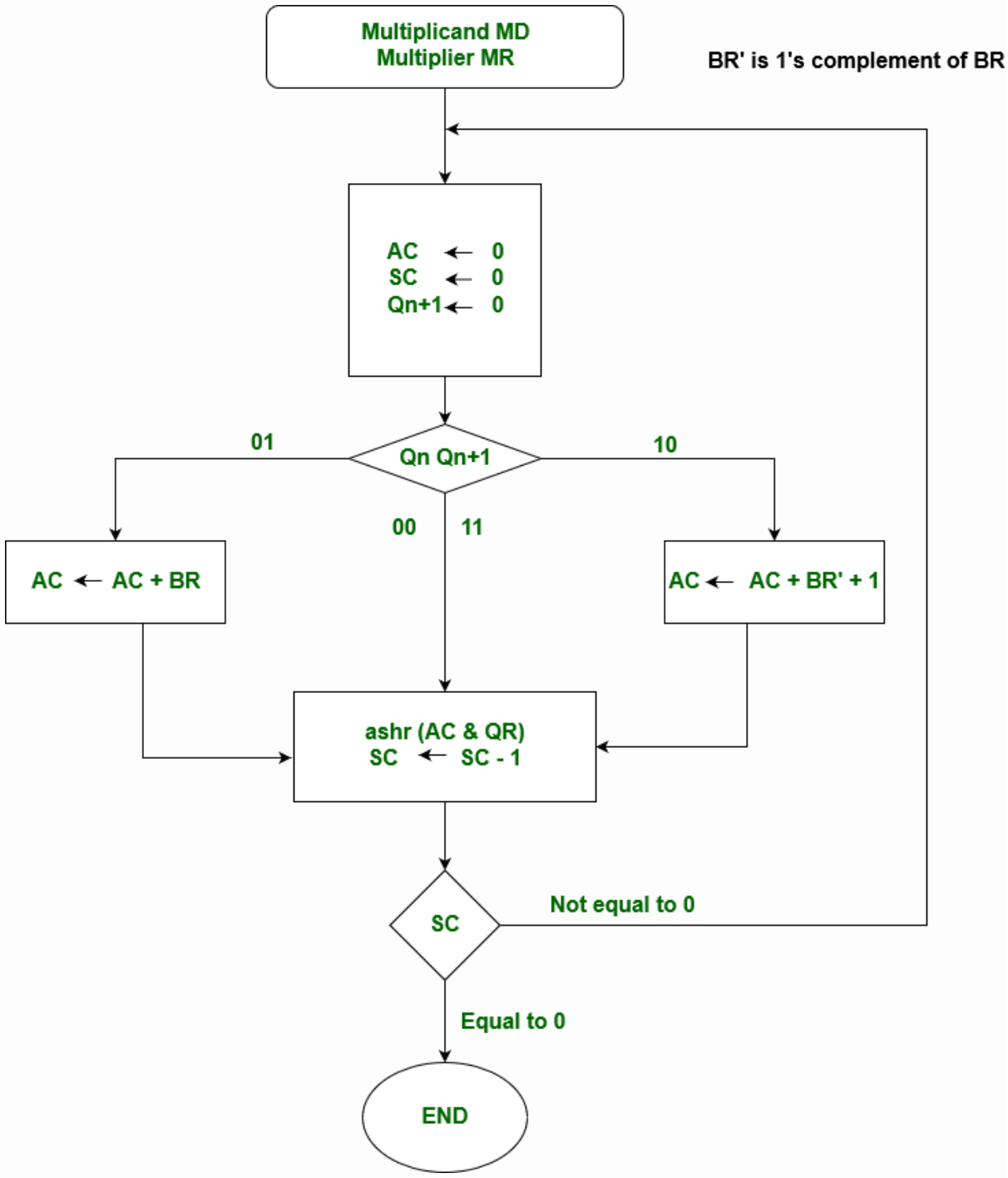
Hardware Implementation of Booths Algorithm – The hardware implementation of the booth algorithm requires the register configuration shown in the figure below.

Booth's Algorithm Flowchart –



We name the register as A, B and Q, AC, BR and QR respectively. Qn designates the least significant bit of multiplier in the register QR. An extra flip-flop Qn+1is appended to QR to facilitate a double inspection of the multiplier. Qn+1 is appended

to QR to facilitate a double inspection of the multiplier. The flowchart for the booth algorithm is shown below.



AC and the appended bit Qn+1 are initially cleared to 0 and the sequence SC is set to a number n equal to the number of bits in the multiplier. The two bits of the multiplier in Qn and Qn+1are inspected. If the two bits are equal to 10, it means that the first 1 in a string has been encountered. This requires subtraction of the multiplicand from the partial product in AC. If the 2 bits are equal to 01, it means that

the first 0 in a string of 0's has n = been encountered. This requires the addition of the multiplicand to the partial product in AC.

When the two bits are equal, the partial product does not change. An overflow cannot occur because the addition and subtraction of the multiplicand follow each other. As a consequence, the 2 numbers that are added always have a opposite signs, a condition that excludes an overflow. The next step is to shift right the partial product and the multiplier (including Qn+1). This is an arithmetic shift right (ashr) operation which AC and QR ti the right and leaves the sign bit in AC unchanged. The sequence counter is decremented and the computational loop is repeated n times.

Example – A numerical example of booth's algorithm is shown below for n = 4. It shows the step by step multiplication of -5 and -7.

```
MD = -5 = 1011, MD = 1011, MD'+1 = 0101
MR = -7 = 1001
The explanation of first step is as follows: Qn+1
AC = 0000, MR = 1001, Qn+1 = 0,  SC = 4
Qn Qn+1 = 10
So, we do AC + (MD)'+1, which gives AC = 0101
On right shifting AC and MR, we get
AC = 0010, MR = 1100 and Qn+1 = 1
```

OPERATION	AC	MR	Qn+1	SC
	0000	1001	0	4
AC + MD' + 1	0101	1001	0	
ASHR	0010	1100	1	3
AC + MR	1101	1100	1	
ASHR	1110	1110	0	2
ASHR	1111	0111	0	1
AC + MD' + 1	0010	0011	1	0

Product is calculated as follows:

```
Product = AC MR
Product = 0010 0011 =  35
```

IEEE Standards:

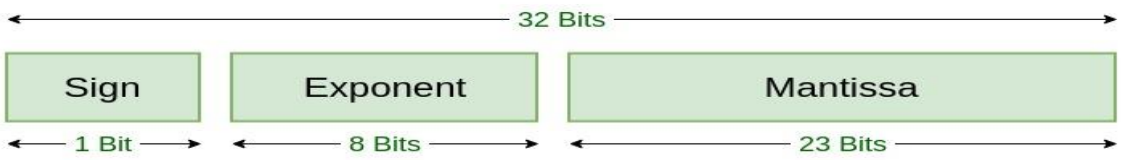
he IEEE Standard for Floating-Point Arithmetic (IEEE) is a technical standard for floating-point computation which was established in 1985 by the **Institute of Electrical and Electronics Engineers (IEEE)**. The standard addressed many problems found in the diverse floating point implementations that made them difficult to use reliably and reduced their portability. IEEE Standard floating point is the most

common representation today for real numbers on computers, including Intel-based PC's, Macs, and most Unix platforms.

There are several ways to represent floating point number but IEEE 754 is the most efficient in most cases. IEEE has 3 basic components:

1. **The Sign of Mantissa –**
   This is as simple as the name. 0 represents a positive number while 1 represents a negative number.
2. **The Biased exponent –**
   The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.
3. **The Normalised Mantisa –**
   The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e. O and 1. So a normalised mantissa is one with only one 1 to the left of the decimal.

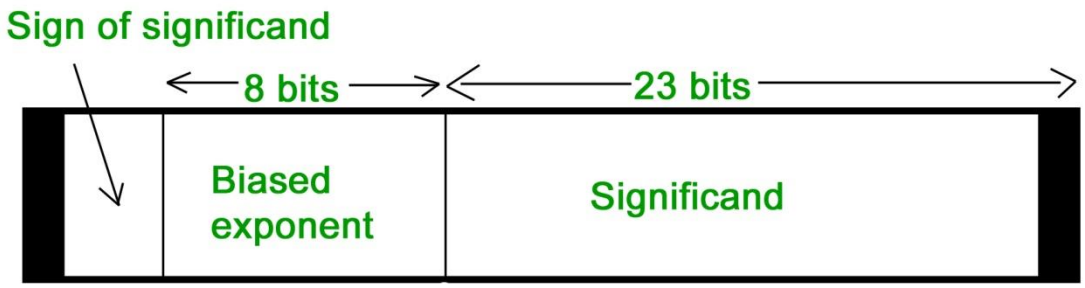
IEEE numbers are divided into two based on the above three components: single precision and double precision.



Single Precision
IEEE 754 Floating-Point Standard

Floating point representation of numbers

- **32-bit representation floating point numbers IEEE standard**



Normalization

- Floating point numbers are usually normalized
- Exponent is adjusted so that leading bit (MSB) of mantissa is 1
- Since it is always 1 there is no need to store it
- Scientific notation where numbers are normalized to give a single digit before the decimal point like in decimal system e.g. 3.123 x 103

For example, we represent 3.625 in 32 bit format.

Changing 3 in binary=11
Changing .625 in binary

```
.625 X 2      1

.25 X 2       0









.5 X 2        1
```

Writing in binary exponent form
3.625=11.101 X 20
On normalizing
11.101 X 20=1.1101 X 21
On biasing exponent = 127 + 1 = 128
$(128)_{10}=(10000000)_2$
For getting significand
Digits after decimal = 1101
Expanding to 23 bit = 11010000000000000000000
Setting sign bit
As it is a positive number, sign bit = 0
Finally we arrange according to representation

```
Sign bit      exponent      significand
```

```
0                10000000    11010000000000000000000
```

- **64-bit representation floating point numbers IEEE standard**

Again we follow the same procedure upto normalization. After that, we add 1023 to bias the exponent.

For example, we represent -3.625 in 64 bit format.
Changing 3 in binary = 11
Changing .625 in binary

```
.625 X 2    1

.25 X 2     0

.5 X 2      1
```

Writing in binary exponent form
3.625 = 11.101 X 20
On normalizing
11.101 X 20 = 1.1101 X 21
On biasing exponent 1023 + 1 = 1024
$(1024)_{10} = (10000000000)_2$
So 11 bit exponent = 10000000000
52 bit significand = 110100000000 ……. making total 52 bits
Setting sign bit = 1 (number is negative)
So, final representation
1 10000000000 110100000000 ……. making total 52 bits by adding further 0's

<u>Converting floating point into decimal</u>
Let's convert a FP number into decimal
1 01111100 11000000000000000000000
The decimal value of an IEEE number is given by the formula:
$(1 -2s) * (1 + f) * 2^{(e - bias)}$
where

- s, f and e fields are taken as decimal here.
- (1 -2s) is 1 or -1, depending upon sign bit 0 and 1
- add an implicit 1 to the significand (fraction field f), as in formula

Again, the bias is either 127 or 1023, for single or double precision respectively.

First convert each individual field to decimal.

- The sign bit s is 1
- The e field contains 01111100 = $(124)_{10}$
- The mantissa is 0.11000 … = $(0.75)_{10}$

Putting these values in formula

$(1 – 2) * (1 + 0.75) * 2124 – 127 = (– 1.75 * 2^{-3}) = – 0.21875$

FLOATING POINT (arithmetic)
ADDITION AND SUBTRACTION

- **FLOATING POINT ADDITION**

To understand floating point addition, first we see addition of real numbers in decimal as same logic is applied in both cases.

For example, we have to add **1.1 * 103** and **50.**

We cannot add these numbers directly. First, we need to align the exponent and then, we can add significand.
After aligning exponent, we get **50 = 0.05 * 103**
Now adding significand, **0.05 + 1.1 = 1.15**
So, finally we get **(1.1 * 103 + 50) = 1.15 * 103**
Here, notice that we shifted **50** and made it **0.05** to add these numbers.

Now let us take example of floating point number addition
We follow these steps to add two numbers:
1. Align the significand
2. Add the significands
3. Normalize the result

Let the two numbers be

x = 9.75
y = 0.5625

Converting them into 32-bit floating point representation,
9.75's representation in 32-bit format = **0 10000010 00111000000000000000000**
0.5625's representation in 32-bit format = **0 01111110 00100000000000000000000**

Now we get the difference of exponents to know how much shifting is required.
$(\mathbf{10000010 – 01111110})_2 = (\mathbf{4})_{10}$
Now, we shift the mantissa of lesser number right side by 4 units.

Mantissa of **0.5625 = 1.00100000000000000000000**
(note that 1 before decimal point is understood in 32-bit representation)
Shifting right by **4** units, we get **0.00010010000000000000000**
Mantissa of **9.75** = **1. 00111000000000000000000**

Adding mantissa of both
0. 00010010000000000000000

+ 1. 00111000000000000000000

1. 01001010000000000000000

In final answer, we take exponent of bigger number
So, final answer consist of :
Sign bit = **0**
Exponent of bigger number = **10000010**
Mantissa = **01001010000000000000000**
32 bit representation of answer = **x + y** = **0 10000010 01001010000000000000000**

- **FLOATING POINT SUBTRACTION**

Subtraction is similar to addition with some differences like we subtract mantissa unlike addition and in sign bit we put the sign of greater number.

Let the two numbers be

x = 9.75
y = – 0.5625

Converting them into 32-bit floating point representation
9.75's representation in 32-bit format = **0 10000010 00111000000000000000000**
– 0.5625's representation in 32-bit format = **1 01111110 00100000000000000000000**

Now, we find the difference of exponents to know how much shifting is required.

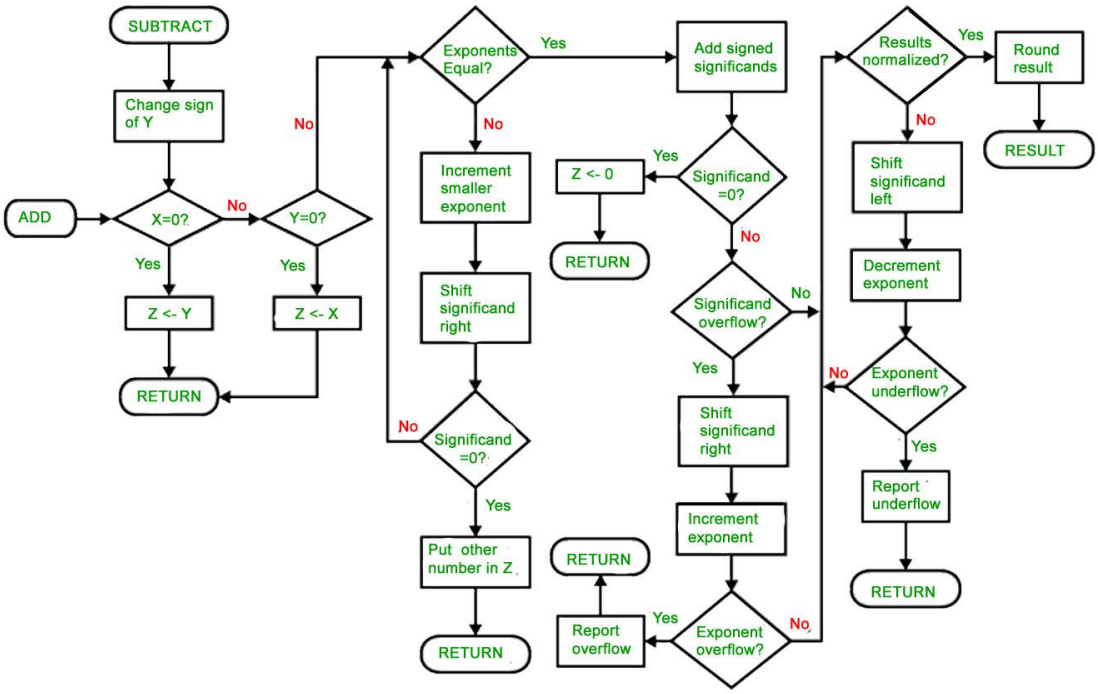
(**10000010 –01111110**)$_2$ = **(4)**$_{10}$

Now, we shift the mantissa of lesser number right side by 4 units.
Mantissa of **– 0.5625 = 1.00100000000000000000000**
(note that 1 before decimal point is understood in 32-bit representation)
Shifting right by **4** units, **0.00010010000000000000000**
Mantissa of **9.75**= **1. 00111000000000000000000**

Subtracting mantissa of both
0. 00010010000000000000000
– 1. 00111000000000000000000

1. 00100110000000000000000

Sign bit of bigger number = **0**
So, finally the answer = **x – y = 0 10000010 00100110000000000000000**



Floating-Point Addition and Subtraction ($Z \leftarrow X \pm Y$)

The accumulator:

In the central processing unit, or CPU, of a computer, the accumulator acts as a special register that stores values and increments of intermediate arithmetic and logic calculations. The accumulator is a temporary memory location that is accessed speedily by the CPU.
The accumulator is the special register of the computer. A register is a special memory location that allows very fast access. Here, the accumulator is a temporary memory location that stores values of all arithmetic and logical calculations that are being carried out by the CPU. The increments of values occur in the accumulator for programming calculations.

The accumulator is the special register of the computer. A register is a special memory location that allows very fast access. Here, the accumulator is a temporary memory location that stores values of all arithmetic and logical calculations that are being carried out by the CPU. The increments of values occur in the accumulator for programming calculations

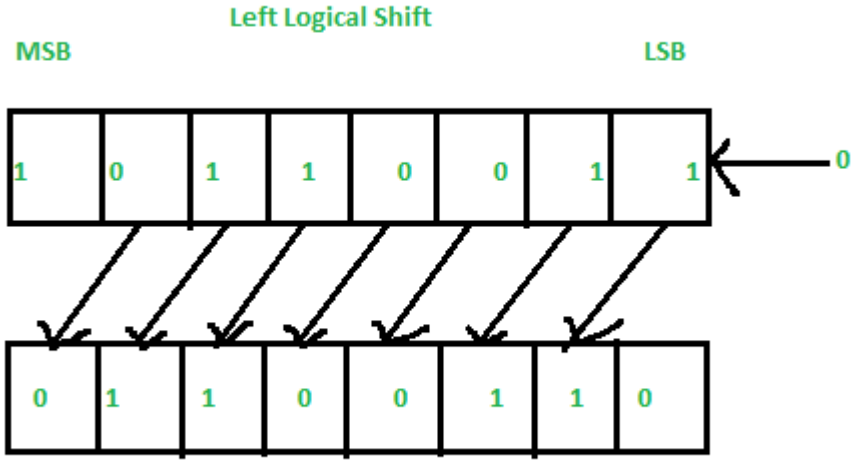
Shifts, Carry and Overflow:

Shift micro-operations are those micro-operations that are used for serial transfer of information. These are also used in conjunction with arithmetic micro-operation, logic micro-operation, and other data-processing operations.

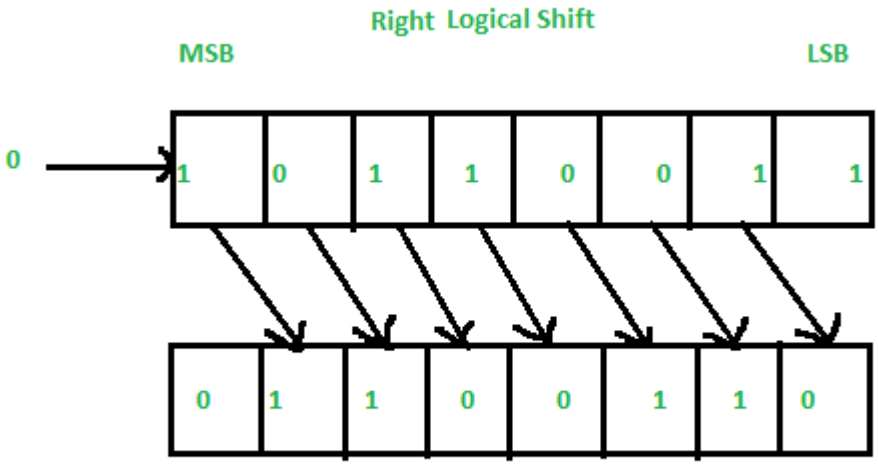
There are three types of shifts micro-operations:

1. Logical :
It transfers the 0 zero through the serial input. We use the symbols shl for logical shift-left and shr for shift-right.

1. **Logical Shift Left –**
   In this shift one position moves each bit to the left one by one. The Empty least significant bit (LSB) is filled with zero (i.e, the serial input), and the most significant bit (MSB) is rejected.

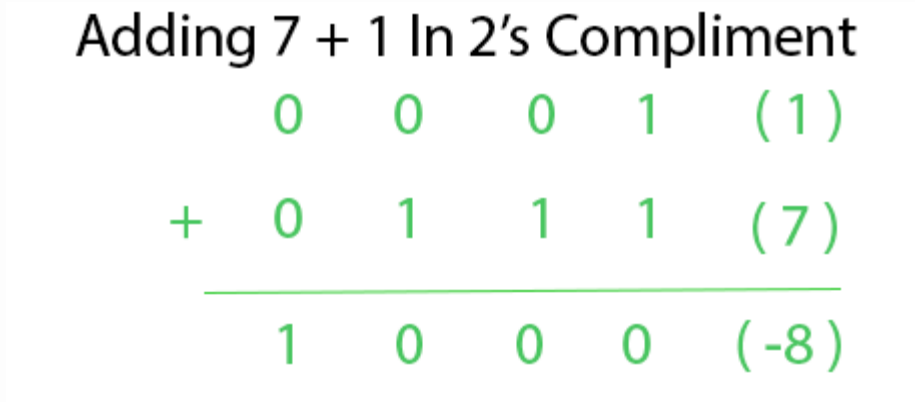


Right Logical Shift –
In this one position moves each bit to the right one by one and the least significant bit(LSB) is rejected and the empty MSB is filled with zero.



In computer Architecture 2's Compliment Number System is widely used. The discussion of overflow here mainly will we with respect to 2's Complimentary System.

N-bit 2's Compliment number System can represent Number from -2^{n-1} to $2^{n-1}-1$
4 Bit can represent numbers from **(-8 to 7)**
5 Bit can represent numbers from **(-16 to 15)** in 2's Complimentary System.
Overflow Occurs with respect to addition when 2 N-bit 2's Compliment Numbers are added and the answer is too large to fit into that N-bit Group.
A computer has N-Bit Fixed registers. Addition of two N-Bit Number will result in max N+1 Bit number. That Extra Bit is stored in carry Flag. But Carry does not always indicate overflow.



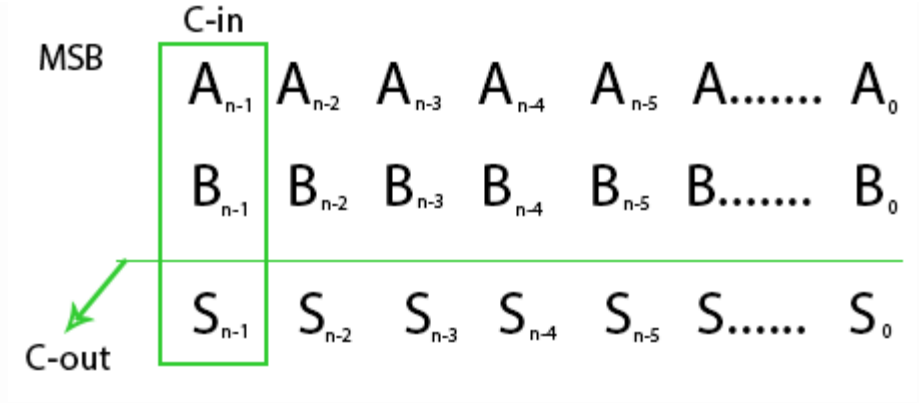
Adding 7 + 1 in 4-Bit must be equal to 8. But 8 cannot be represented with 4 bit 2's complement number as it is out of range. Two Positive numbers were added and the answer we got is negative (-8). Here Carry is also 0. It is normally left to the programmer to detect overflow and deal with this situation.

Overflow Detection –

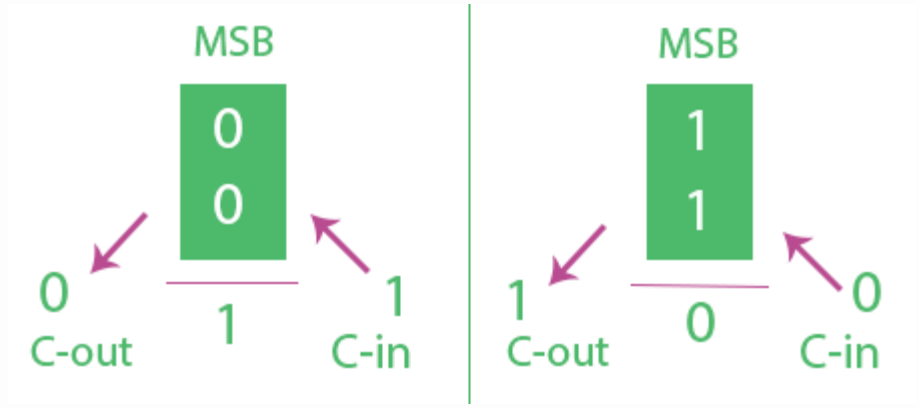
Overflow occurs when:

1. Two negative numbers are added and an answer comes positive or
2. Two positive numbers are added and an answer comes as negative.

So overflow can be detected by checking Most Significant Bit(MSB) of two operands and answer. But Instead of using 3-bit Comparator Overflow can also be detected using 2 Bit Comparator just by checking Carry-in(C-in) and Carry-Out(C-out) from MSB's. Consider N-Bit Addition of 2's Compliment number.



Overflow Occurs when C-in $\neq$ C-out. Above expression for overflow can be explained from below Analysis.

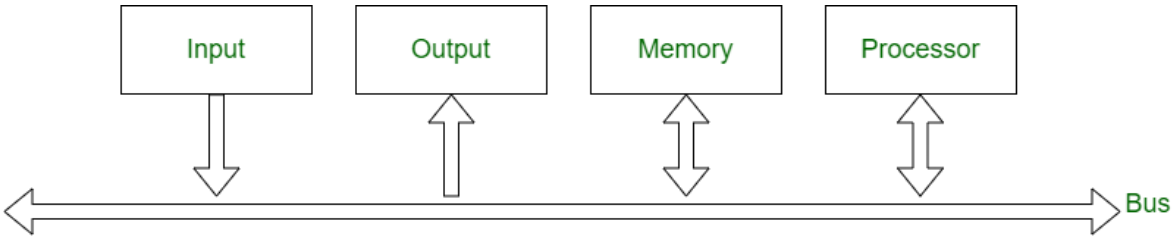


In first Figure the MSB of two numbers are 0 which means they are positive. Here if **C-in is 1** we get answer's MSB as 1 means answer is negative (Overflow) and **C-out as 0.** C-in $\neq$ C-out hence overflow.
In second Figure the MSB of two numbers are 1 which means they are negative. Here if **C-in is 0** we get answer MSB as 0 means answer is positive(Overflow) and **C-out as 1.** C-in $\neq$ C-out hence overflow.
Readers can also try out other combination of c-in c-out and MSB's to check overflow.
So Carry-in and Carry-out at MSB's are enough to detect Overflow.

Above XOR Gate can be used to detect overflow.

CPU with Single BUS:

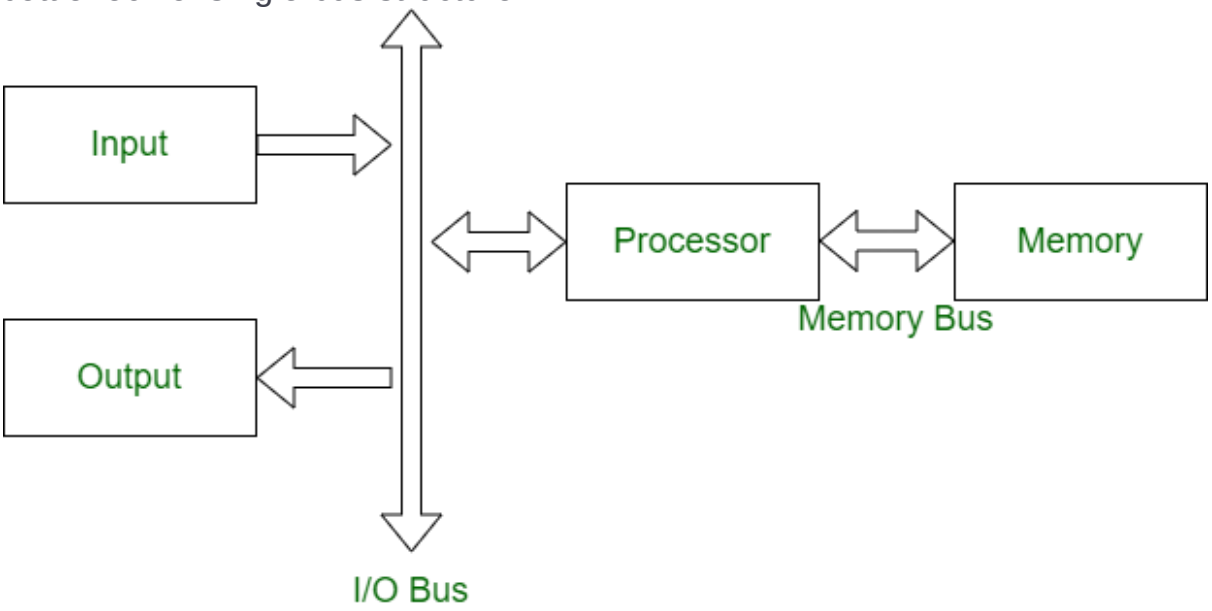
1. **Single Bus Structure :**
   In single bus structure, one common bus used to communicate between peripherals and microprocessor. It has disadvantages due to use of one common bus.



Single Bus Structure

---

2. **Double Bus Structure :**
   In double bus structure, one bus is used to fetch instruction while other is used to fetch data, required for execution. It is to overcome the bottleneck of single bus structure.



Double Bus Structure

Types of Operands, Types of Operations

Instruction Formats (Zero, One, Two and Three Address Instruction)

Computer perform task on the basis of instruction provided. A instruction in computer comprises of groups called fields. These field contains different information as for computers every thing is in 0 and 1 so each field has different significance on the basis of which a CPU decide what so perform. The most common fields are:

- Operation field which specifies the operation to be performed like addition.
- Address field which contain the location of operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

A instruction is of various length depending upon the number of addresses it contain. Generally CPU organization are of three types on the basis of number of address fields:

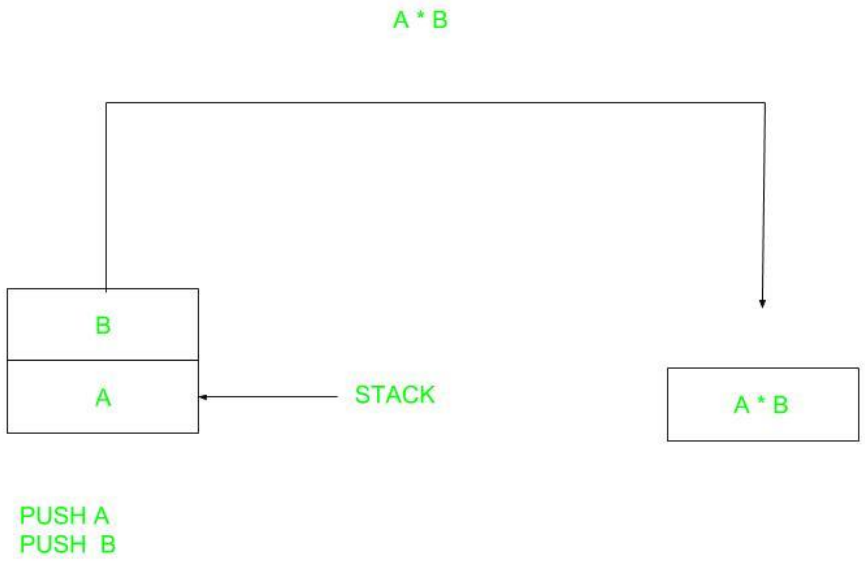
1. Single Accumulator organization
2. General register organization
3. Stack organization

In first organization operation is done involving a special register called accumulator. In second on multiple registers are used for the computation purpose. In third organization the work on stack basis operation due to which it does not contain any address field. It is not necessary that only a single organization is is applied a blend of various organization is mostly what we see generally.

On the basis of number of address instruction are classified as:

Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

1. **Zero Address Instructions –**



A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

```
Expression: X = (A+B)*(C+D)
Postfixed : X = AB+CD+*
TOP means top of stack
M[X] is any memory location
```

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B

PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

2. **One Address Instructions –**
This use a implied ACCUMULATOR register for data manipulation.One operand is in accumulator and other is in register or memory location.Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.


```
Expression: X = (A+B)*(C+D)
AC is accumulator
M[] is any memory location
M[T] is temporary location
```

LOAD	A	AC = M[A]
ADD	B	AC = AC + 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

3. **Two Address Instructions –**
This is common in commercial computers.Here two address can be specified in the instruction.Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

Here destination address can also contain operand.

```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
```

```
M[] is any memory location
```

MOV	R1, A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]
MOV	R2, C	R2 = C
ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

4. **Three Address Instructions –**
This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.


```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

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

Addressing Modes:

Addressing modes specifies the way, the effective address of an operand is represented in the instruction. Some addressing mode efficiently allows referring to a large range of area like a linear array of addresses and list of addresses. Addressing mode describes a **flexible** and **efficient** way to define complex **effective address**.

Generally, the programs are written in a high-level language, as it is a convenient way to define the variables and operations that the programmer needs to perform on the variables. Later, this program is compiled to generate the machine code. The machine code has low-level instructions.

The low-level instruction has **opcode** and **operands**. Addressing mode has nothing to do with the opcode part. It focuses on presenting the operand's address in the instructions.

We have the list below showing the various kind of addressing modes:

Types of Addressing Modes

1. Register Addressing Mode
2. Direct Addressing Mode
3. Immediate Addressing Mode
4. Register Indirect Addressing Mode
5. Index Addressing Mode
6. Auto Increment /Decrement Mode
7. Relative Addressing Mode

Instruction Formats:

Instruction format

1. **Instruction format** describes the internal structures (layout design) of the bits of an instruction, in terms of its constituent parts.
2. An **Instruction format** must include an opcode, and address is dependent on an availability of particular operands.
3. The format can be implicit or explicit which will indicate the addressing mode for each operand.
4. Designing of an **Instruction format** is very complex. As we know a computer uses a variety of instructional. There are many designing issues which affect the instructional design, some of them are given are below:
    - **Instruction length:** It is a most basic issue of the format design. A longer will be the instruction it means more time is needed to fetch the instruction.
    - **Memory size:** If larger memory range is to be addressed then more bits will be required in the address field.
    - **Memory organization:** If the system supports the virtual memory then memory range which needs to be addressed by the instruction, is larger than the physical memory.
    - **Memory transfer length:** Instruction length should be equal to the data bus length or it should be multiple of it.
5. **Instruction formats** are classified into 5 types based on the type of the CPU organization. CPU organization is divided into three types based on the availability of the ALU operands, which are as follows here:

UNIT II

Processor Organization:

Parallelism and Computer arithmetic:

Parallelism in Computer Arithmetic: A Historical Perspective Many early parallel processing breakthroughs emerged from the quest for faster and higher-throughput arithmetic operations. Additionally, the influence of arithmetic techniques on parallel computer performance can be seen in diverse areas such the bit-serial arithmetic units of early massively parallel SIMD computers, pipelining and pipeline chaining in vector machines, design of floating-point standards to ensure the accuracy and portability of numerically-intensive programs, and prominence of GPUs in today's top-of-the-line supercomputers.

Computer arithmetic associatively:

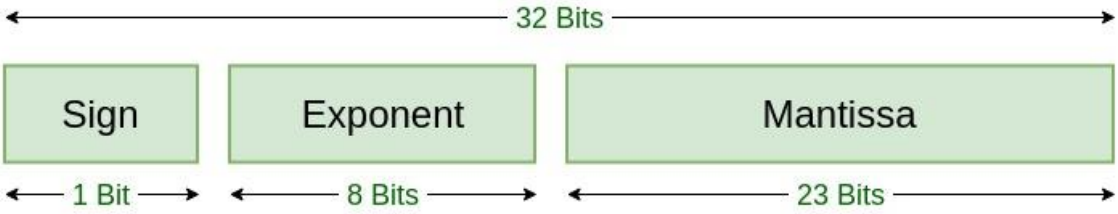
Computers Operations on integers
Addition and subtraction
Multiplication and division
Dealing with overflow
Floating-point real numbers

Floating point in 8086

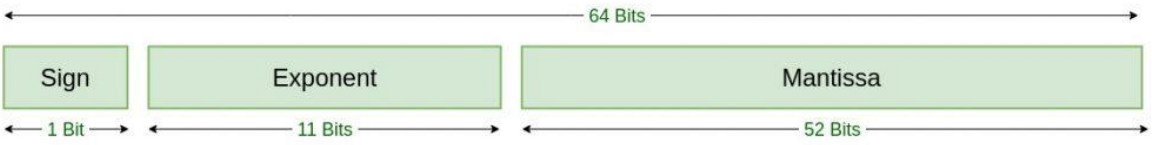
The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation which was established in 1985 by the **Institute of Electrical and Electronics Engineers (IEEE)**. The standard addressed many problems found in the diverse floating point implementations that made them difficult to use reliably and reduced their portability. IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macs, and most Unix platforms.
There are several ways to represent floating point number but IEEE 754 is the most efficient in most cases. IEEE 754 has 3 basic components:

1. **The Sign of Mantissa –**
   This is as simple as the name. 0 represents a positive number while 1 represents a negative number.
2. **The Biased exponent –**
   The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.
3. **The Normalised Mantissa –**
   The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e. O and 1. So a normalised mantissa is one with only one 1 to the left of the decimal.

IEEE 754 numbers are divided into two based on the above three components: single precision and double precision.



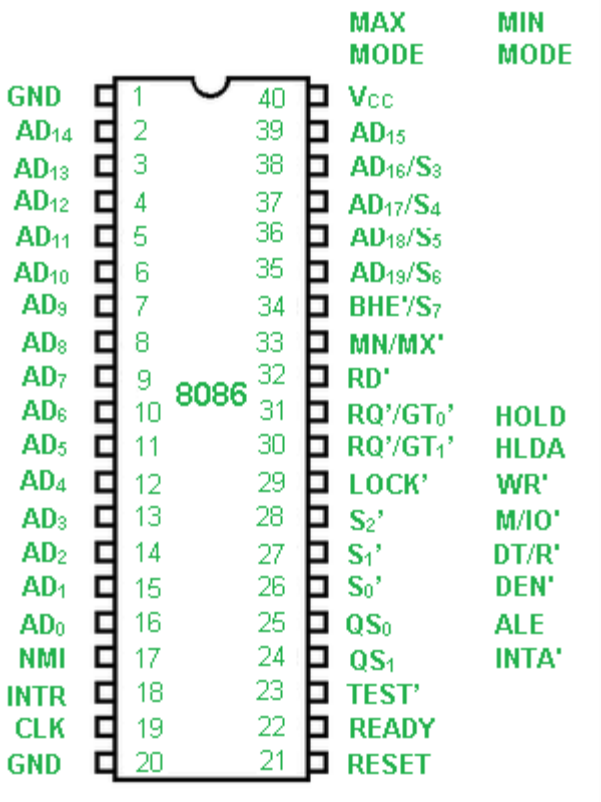
Single Precision
IEEE 754 Floating-Point Standard



Double Precision
IEEE 754 Floating-Point Standard

Programmer Model of 8086 :

Pin diagram of 8086 microprocessor is as given below:



Intel 8086 is a 16-bit HMOS microprocessor. It is available in 40 pin DIP chip. It uses a 5V DC supply for its operation. The 8086 uses 20-line address bus. It has a 16-line data bus. The 20 lines of the address bus operate in multiplexed mode. The 16-low order address bus lines have been multiplexed with data and 4 high-order address bus lines have been multiplexed with status signals.

AD0-AD15 : Address/Data bus. These are low order address bus. They are multiplexed with data. When AD lines are used to transmit memory address the symbol A is used instead of AD, for example A0-A15. When data are transmitted over AD lines the symbol D is used in place of AD, for example D0-D7, D8-D15 or D0-D15.
A16-A19 : High order address bus. These are multiplexed with status signals.
S2, S1, S0 : Status pins. These pins are active during T4, T1 and T2 states and is returned to passive state (1,1,1 during T3 or Tw (when ready is inactive). These are used by the 8288 bus controller for generating all the memory and I/O operation) access control signals. Any change in S2, S1, S0 during T4 indicates the beginning of a bus cycle.

S2	S1	S0	Characteristics
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive state

A16/S3, A17/S4, A18/S5, A19/S6 : The specified address lines are multiplexed with corresponding status signals.

A17/S4	A16/S3	Function
0	0	Extra segment access
0	1	Stack segment access
1	0	Code segment access
1	1	Data segment access

BHE'/S7 : Bus High Enable/Status. During T1 it is low. It is used to enable data onto the most significant half of data bus, D8-D15. 8-bit device connected to upper half of the data bus use BHE (Active Low) signal. It is multiplexed with status signal S7. S7 signal is available during T2, T3 and T4.
RD': This is used for read operation. It is an output signal. It is active when low.
READY : This is the acknowledgement from the memory or slow device that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the microprocessor. The signal is active high(1).
INTR : Interrupt Request. This is triggered input. This is sampled during the last clock cycles of each instruction for determining the availability of the request. If any interrupt request is found pending, the processor enters the interrupt acknowledge cycle. This can be internally masked after resulting the interrupt enable flag. This signal is active high(1) and has been synchronized internally.

NMI : Non maskable interrupt. This is an edge triggered input which results in a type II interrupt. A subroutine is then vectored through an interrupt vector lookup table which is located in the system memory. NMI is non-maskable internally by software. A transition made from low(0) to high(1) initiates the interrupt at the end of the current instruction. This input has been synchronized internally.
INTA : Interrupt acknowledge. It is active low(0) during T2, T3 and Tw of each interrupt acknowledge cycle.
MN/MX' : Minimum/Maximum. This pin signal indicates what mode the processor will operate in.
RQ'/GT1', RQ'/GT0' : Request/Grant. These pins are used by local bus masters used to forc the microprocessor to release the local bus at the end of the microprocessor's current bus cycle. Each of the pin is bi-directional. RQ'/GT0' have higher priority than RQ'/GT1'.
LOCK' : Its an active low pin. It indicates that other system bus masters have not been allowed to gain control of the system bus while LOCK' is active low(0). The LOCK signal will be active until the completion of the next instruction.
TEST' : This examined by a 'WAIT' instruction. If the TEST pin goes low(0), execution will continue, else the processor remains in an idle state. The input is internally synchronized during each of the clock cycle on leading edge of the clock.
CLK : Clock Input. The clock input provides the basic timing for processing operation and bus control activity. Its an asymmetric square wave with a 33% duty cycle.
RESET : This pin requires the microprocessor to terminate its present activity immediately. The signal must be active high(1) for at least four clock cycles.
Vcc : Power Supply(+5V D.C.)
GND : Ground
QS1,QS0 : Queue Status. These signals indicate the status of the internal 8086 instruction queue according to the table shown below

QS1	QS0	Status
0	0	No operation
0	1	First byte of op code from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

DT/R : Data Transmit/Receive. This pin is required in minimum systems, that want to use an 8286 or 8287 data bus transceiver. The direction of data flow is controlled through the transceiver.
DEN : Data enable. This pin is provided as an output enable for the 8286/8287 in a minimum system which uses transceiver. DEN is active low(0) during each memory and input-output access and for INTA cycles.
HOLD/HOLDA : HOLD indicates that another master has been requesting a local bus .This is an active high(1). The microprocessor receiving the HOLD request will issue HLDA (high) as an acknowledgement in the middle of a T4 or T1 clock cycle.
ALE : Address Latch Enable. ALE is provided by the microprocessor to latch the address into the 8282 or 8283 address latch. It is an active high(1) pulse during T1 of any bus cycle. ALE signal is never floated, is always integer.

Register organisation 8086 registers :

General purpose registers are used to store temporary data within the microprocessor. There are 8 general purpose registers in 8086 microprocessor.

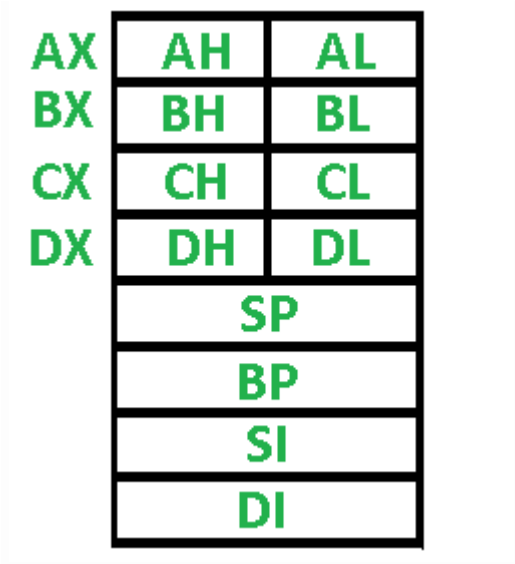


Figure – General purpose registers

1. **AX –** This is the accumulator. It is of 16 bits and is divided into two 8-bit registers AH and AL to also perform 8-bit instructions.
   It is generally used for arithmetical and logical instructions but in 8086 microprocessor it is not mandatory to have accumulator as the destination operand.

   Example:

   ```
   ADD AX, AX (AX = AX + AX)
   ```

2. **BX –** This is the base register. It is of 16 bits and is divided into two 8-bit registers BH and BL to also perform 8-bit instructions.
   It is used to store the value of the offset.
   Example:

   ```
   MOV BL, [500] (BL = 500H)
   ```

3. **CX –** This is the counter register. It is of 16 bits and is divided into two 8-bit registers CH and CL to also perform 8-bit instructions.
   It is used in looping and rotation.

   Example:

   ```
   MOV CX, 0005
   LOOP
   ```

4. **DX –** This is the data register. It is of 16 bits and is divided into two 8-bit registers DH and DL to also perform 8-bit instructions.
   It is used in multiplication an input/output port addressing.
   Example:

   ```
   MUL BX (DX, AX = AX * BX)
   ```

5. **SP –** This is the stack pointer. It is of 16 bits.
   It points to the topmost item of the stack. If the stack is empty the stack pointer will be (FFFE)H. It's offset address relative to stack segment.
6. **BP –** This is the base pointer. It is of 16 bits.
   It is primary used in accessing parameters passed by the stack. It's offset address relative to stack segment.

7. **SI –** This is the source index register. It is of 16 bits.
   It is used in the pointer addressing of data and as a source in some string related operations. It's offset is relative to data segment.
8. **DI –** This is the destination index register. It is of 16 bits.
   It is used in the pointer addressing of data and as a destination in some string related operations.It's offset is relative to extra segment.

Micro instruction:

Logical instructions

Logical instructions are the instructions which perform basic logical operations such as AND, OR, etc. In 8086 microprocessor, the destination operand need not be the accumulator.

Following is the table showing the list of logical instructions:

OPCODE	OPERAND	DESTINATION	EXAMPLE
AND	D, S	D = D AND S	AND AX, 0010
OR	D, S	D = D OR S	OR AX, BX
NOT	D	D = NOT of D	NOT AL
XOR	D, S	D = D XOR S	XOR AL, BL
TEST	D, S	performs bit-wise AND operation and affects the flag registor	TEST [0250], 06
SHR	D, C	shifts each bit in D to the right C times and 0 is stored at MSB position	SHR AL, 04
SHL	D, C	shifts each bit in D to the left C times and 0 is stored at LSB position	SHL AX, BL
ROR	D, C	rotates all bits in D to the right C times	ROR BL, CL
ROL	R, C	rotates all bits in D to the left C times	ROL BX, 06
RCR	D, C	rotates all bits in D to the right along with carry flag C times	RCR BL, CL
RCL	R, C	rotates all bits in D to the left along with carry flag C times	RCL BX, 06

Here D stands for destination, S stands for source and C stands for count. They can either be register, data or memory address.

Process control instructions

Process control instructions are the instructions which control the processor's action by setting(1) or resetting(0) the values of flag registers.

Following is the table showing the list of process control instructions:

OPCODE	OPERAND	EXPLPANATION	EXAMPLE
STC	none	sets carry flag to 1	STC
CLC	none	resets carry flag to 0	CLC
CMC	none	compliments the carry flag	CMC

OPCODE	OPERAND	EXPLPANATION	EXAMPLE
STD	none	sets directional flag to 1	STD
CLD	none	resets directional flag to 0	CLD
STI	none	sets the interrupt flag to 1	STI
CLI	none	resets the interrupt flag to 0	CLI

Arithmetic instructions

Arithmetic Instructions are the instructions which perform basic arithmetic operations such as addition, subtraction and a few more. Unlike in 8085 microprocessor, in 8086 microprocessor the destination operand need not be the accumulator.

Following is the table showing the list of arithmetic instructions:

OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD AX, [2050]
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
SUB	D, S	D = D – S	SUB AX, [SI]
SBB	D, S	D = D – S – prev. carry	SBB [2050], 0050
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
INC	D	D = D + 1	INC AX
DEC	D	D = D – 1	DEC [2050]
CBW	none	converts signed byte to word	CBW
CWD	none	converts signed byte to double word	CWD
NEG	D	D = 2’s compliment of D	NEG AL
DAA	none	decimal adjust accumulator	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM

OPCODE	OPERAND	EXPLANATION	EXAMPLE
AAD	none	ASCII adjust accumulator after division	AAD

Here D stands for destination and S stands for source. D and S can either be register, data or memory address.

The Instruction cycle :

An instruction cycle, also known as fetch-decode-execute cycle is the basic operational process of a computer. This process is repeated continuously by CPU from boot up to shut down of computer.

Following are the steps that occur during an instruction cycle:

1. Fetch the Instruction

The instruction is fetched from memory address that is stored in PC(Program Counter) and stored in the instruction register IR. At the end of the fetch operation, PC is incremented by 1 and it then points to the next instruction to be executed.

2. Decode the Instruction

The instruction in the IR is executed by the decoder.

3. Read the Effective Address

If the instruction has an indirect address, the effective address is read from the memory. Otherwise operands are directly read in case of immediate operand instruction.

4. Execute the Instruction

The Control Unit passes the information in the form of control signals to the functional unit of CPU. The result generated is stored in main memory or sent to an output device.

The cycle is then repeated by fetching the next instruction. Thus in this way the instruction cycle is repeated continuously.

Addressing modes :

Prerequisite – Addressing modes, Addressing modes in 8085 microprocessor

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
```

4. **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
```

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

```
9. MOV AX, [DISP]
```

```
MOV AX, [0500]
```

10. **Register indirect mode –** In this addressing mode the effective address is in SI, DI or BX.
    Example:

```
11. MOV AX, [DI]
12. ADD AL, [BX]
```

```
MOV AX, [SI]
```

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

```
14. 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]
```

15. **Indexed mode –** In this type of addressing mode the effective address is sum of index register and displacement.
    Example:

```
16. MOV AX, [SI+2000]
```

```
MOV AL, [DI+3000]
```

17. **Based mode –** In this the effective address is the sum of base register and displacement.
    Example:

```
MOV AL, [BP+ 0100]
```

18. **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]
```

19. **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:

```
20. MOVS B
```

```
MOVS W
```

21. **Input/Output mode –** This addressing mode is related with input output operations.
    Example:

```
22. IN A, 45
```

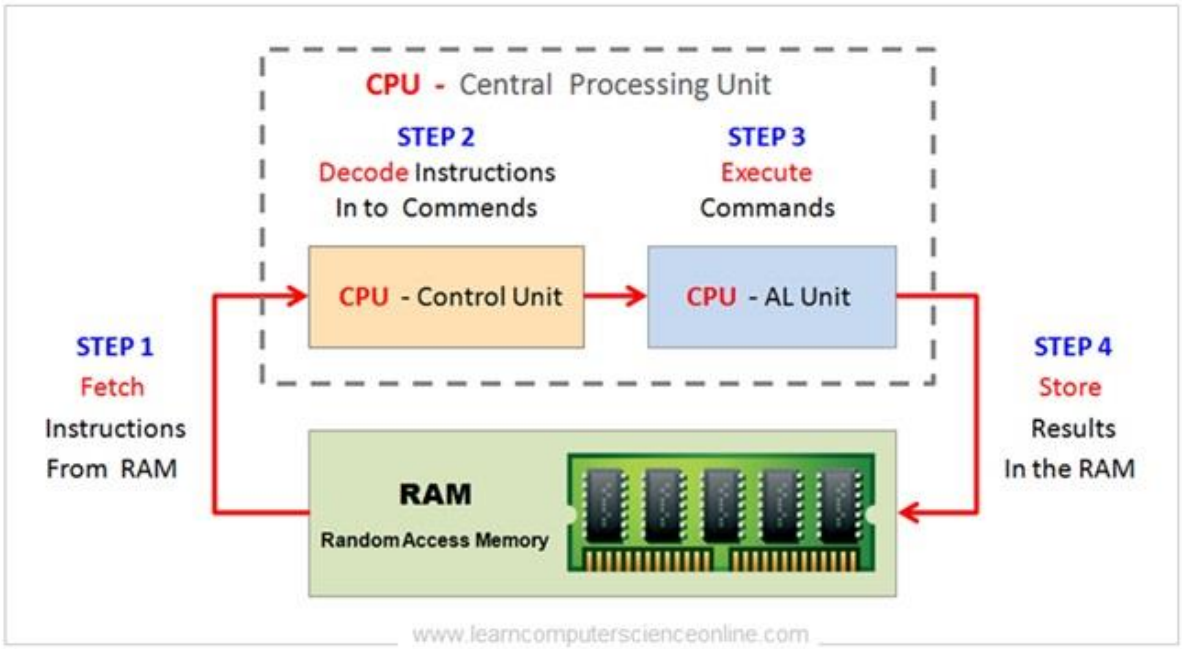
```
OUT A, 50
```

23. **Relative mode –**
    In this the effective address is calculated with reference to instruction pointer.
    Example:

```
24. JNZ 8 bit address
```

```
IP=IP+8 bit address
```

Functional Requirements Control of the CPU:



Execution of a complete instruction:

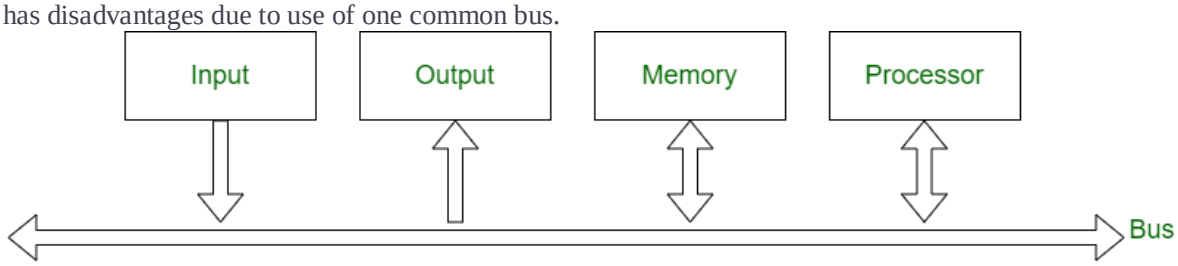
Execution of a Complete Instructions:

1. Fetch information from memory to CPU.
2. Store information to CPU register to memory.
3. Transfer of data between CPU registers.
4. **Perform** arithmetic or logic operation and store the result in CPU registers.

Difference between Single Bus Structure and Double Bus Structure

1. Single Bus Structure :

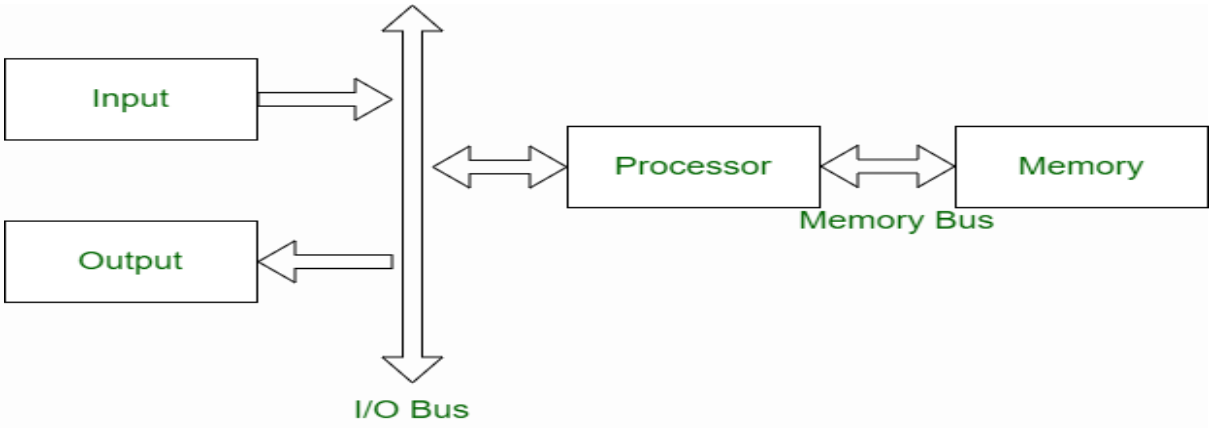
In single bus structure, one common bus used to communicate between peripherals and microprocessor. It has disadvantages due to use of one common bus.



Single Bus Structure

2. Double Bus Structure :

In double bus structure, one bus is used to fetch instruction while other is used to fetch data, required for execution. It is to overcome the bottleneck of single bus structure.

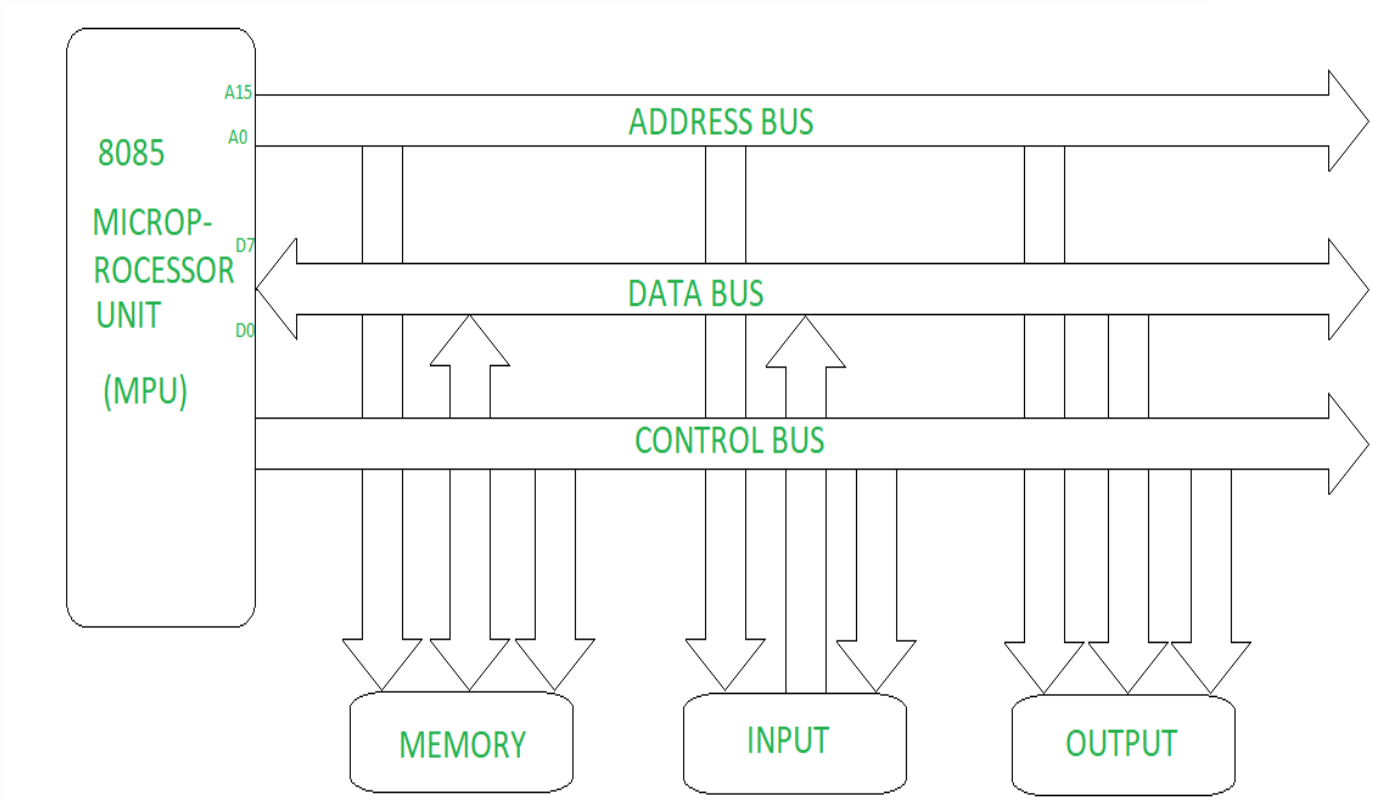


Double Bus Structure

Multiple Bus organization

Bus is a group of conducting wires which carries information, all the peripherals are connected to microprocessor through Bus.

Diagram to represent bus organization system of 8085 Microprocessor.



Bus organization system of 8085 Microprocessor

There are three types of buses.

1. **Address bus –**
   It is a group of conducting wires which carries address only.Address bus is unidirectional because data flow in one direction, from microprocessor to memory or from microprocessor to Input/output devices (That is, Out of Microprocessor).

Length of Address Bus of 8085 microprocessor is 16 Bit (That is, Four Hexadecimal Digits), ranging from 0000 H to FFFF H, (H denotes Hexadecimal). The microprocessor 8085 can transfer maximum 16 bit address which means it can address 65, 536 different memory location.

The Length of the address bus determines the amount of memory a system can address.Such as a system with a 32-bit address bus can address 2^32 memory locations.If each memory location holds one byte, the addressable memory space is 4 GB.However, the actual amount of memory that can be accessed is usually much less than this theoretical limit due to chipset and motherboard limitations.

2. **Data bus –**
   It is a group of conducting wires which carries Data only.Data bus is bidirectional because data flow in both directions, from microprocessor to memory or Input/Output devices and from memory or Input/Output devices to microprocessor.
   Length of Data Bus of 8085 microprocessor is 8 Bit (That is, two Hexadecimal Digits), ranging from 00 H to FF H. (H denotes Hexadecimal).

   When it is write operation, the processor will put the data (to be written) on the data bus, when it is read operation, the memory controller will get the data from specific memory block and put it into the data bus.

   The width of the data bus is directly related to the largest number that the bus can carry, such as an 8 bit bus can represent 2 to the power of 8 unique values, this equates to the number 0 to 255.A 16 bit bus can carry 0 to 65535.

3. **Control bus –**
   It is a group of conducting wires, which is used to generate timing and control signals to control all the associated peripherals, microprocessor uses control bus to process data, that is what to do with selected memory location. Some control signals are:

- Memory read
- Memory write
- I/O read
- I/O Write
- Opcode fetch

If one line of control bus may be the read/write line.If the wire is low (no electricity flowing) then the memory is read, if the wire is high (electricity is flowing) then the memory is written.

Branching:

Branching instructions in 8085 microprocessor

Branching instructions refer to the act of switching execution to a different instruction sequence as a result of executing a branch instruction.

The three types of branching instructions are:

1. Jump (unconditional and conditional)
2. Call (unconditional and conditional)
3. Return (unconditional and conditional)

1. Jump Instructions – The jump instruction transfers the program sequence to the memory address given in the operand based on the specified flag. Jump instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

(a) Unconditional Jump Instructions: Transfers the program sequence to the described memory address.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JMP	address	Jumps to the address	JMP 2050

(b) Conditional Jump Instructions: Transfers the program sequence to the described memory address only if the condition in satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JC	address	Jumps to the address if carry flag is 1	JC 2050
JNC	address	Jumps to the address if carry flag is 0	JNC 2050
JZ	address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	address	Jumps to the address if parity flag is 1	JPE 2050
JPO	address	Jumps to the address if parity flag is 0	JPO 2050
JM	address	Jumps to the address if sign flag is 1	JM 2050
JP	address	Jumps to the address if sign flag 0	JP 2050

2. Call Instructions – The call instruction transfers the program sequence to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack. Call instructions are 2 types: Unconditional Call Instructions and Conditional Call Instructions.

(a) Unconditional Call Instructions: It transfers the program sequence to the memory address given in the operand.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CALL	address	Unconditionally calls	CALL 2050

(b) Conditional Call Instructions: Only if the condition is satisfied, the instructions executes.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CC	address	Call if carry flag is 1	CC 2050
CNC	address	Call if carry flag is 0	CNC 2050
CZ	address	Calls if zero flag is 1	CZ 2050
CNZ	address	Calls if zero flag is 0	CNZ 2050
CPE	address	Calls if carry flag is 1	CPE 2050
CPO	address	Calls if carry flag is 0	CPO 2050
CM	address	Calls if sign flag is 1	CM 2050
CP	address	Calls if sign flag is 0	CP 2050

3. Return Instructions – The return instruction transfers the program sequence from the subroutine to the calling program. Jump instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

(a) Unconditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RET	none	Return from the subroutine unconditionally	RET

(b) Conditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program only is the condition is satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RC	none	Return from the subroutine if carry flag is 1	RC
RNC	none	Return from the subroutine if carry flag is 0	RNC
RZ	none	Return from the subroutine if zero flag is 1	RZ
RNZ	none	Return from the subroutine if zero flag is 0	RNZ
RPE	none	Return from the subroutine if parity flag is 1	RPE
RPO	none	Return from the subroutine if parity flag is 0	RPO
RM	none	Returns from the subroutine if sign flag is 1	RM
RP	none	Returns from the subroutine if sign flag is 0	RP

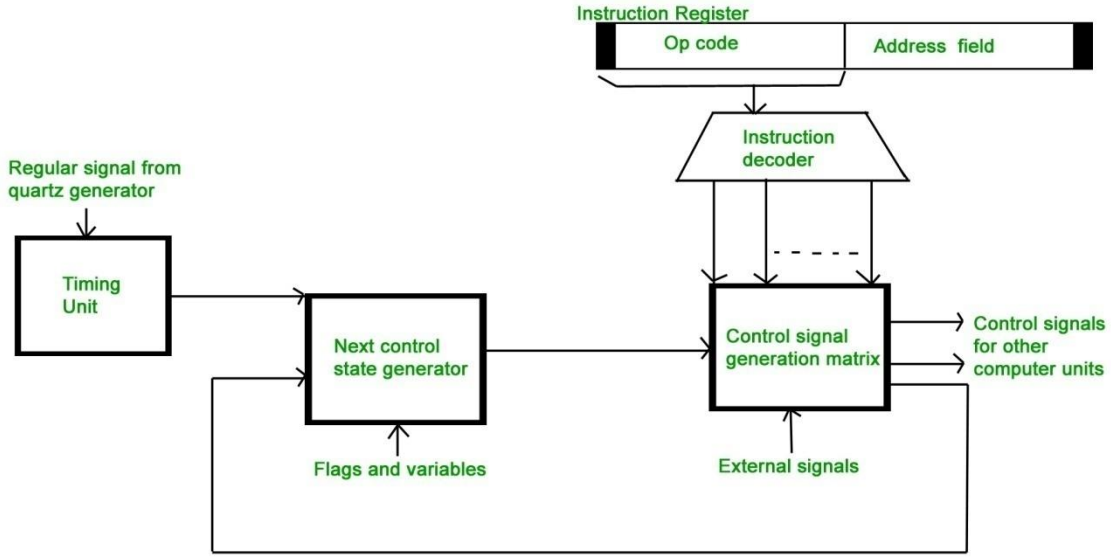
Hardwired Control Unit, Micro-Programmed Control.

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. There are two approaches used for generating the control signals in proper sequence as Hardwired Control unit and Micro-programmed control unit.

Hardwired Control Unit –

The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs. The outputs of the state machine are the control signals. The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as “hardwired”.

- Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.
- Hardwired control is faster than micro-programmed control.
- A controller that uses this approach can operate at high speed.
- RISC architecture is based on hardwired control unit

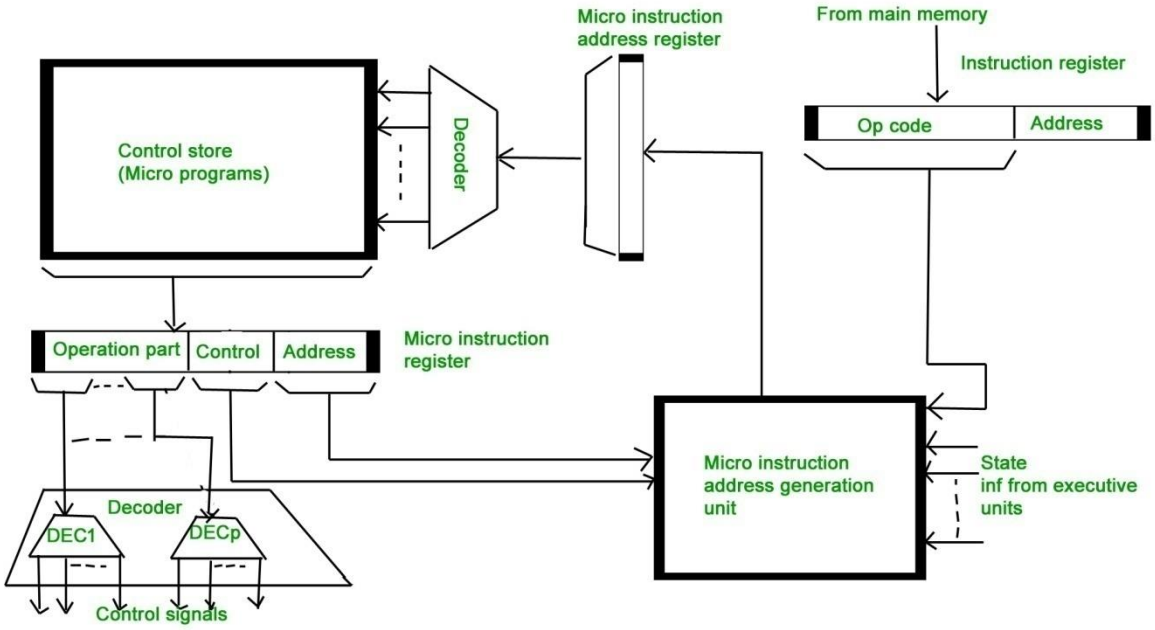


Micro-programmed Control Unit –

- The control signals associated with operations are stored in special memory units inaccessible by the programmer as Control Words.
- Control signals are generated by a program are similar to machine language programs.
- Micro-programmed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory.

Some Important Terms –

1. **Control Word :** A control word is a word whose individual bits represent various control signals.
2. **Micro-routine :** A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.
3. **Micro-instruction :** Individual control words in this micro-routine are referred to as microinstructions.
4. **Micro-program :** A sequence of micro-instructions is called a micro-program, which is stored in a ROM or RAM called a Control Memory (CM).
5. **Control Store :** the micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the Control Store.



Types of Micro-programmed Control Unit – Based on the type of Control Word stored in the Control Memory (CM), it is classified into two types :

1. Horizontal Micro-programmed control Unit :

The control signals are represented in the decoded binary format that is 1 bit/CS. Example: If 53 Control signals are present in the processor than 53 bits are required. More than 1 control signal can be enabled at a time.

- It supports longer control word.
- It is used in parallel processing applications.
- It allows higher degree of parallelism. If degree is n, n CS are enabled at a time.
- It requires no additional hardware(decoders). It means it is faster than Vertical Microprogrammed.
- It is more flexible than vertical microprogrammed

2. Vertical Micro-programmed control Unit :

The control signals re represented in the encoded binary format. For N control signals- Log2(N) bits are required.

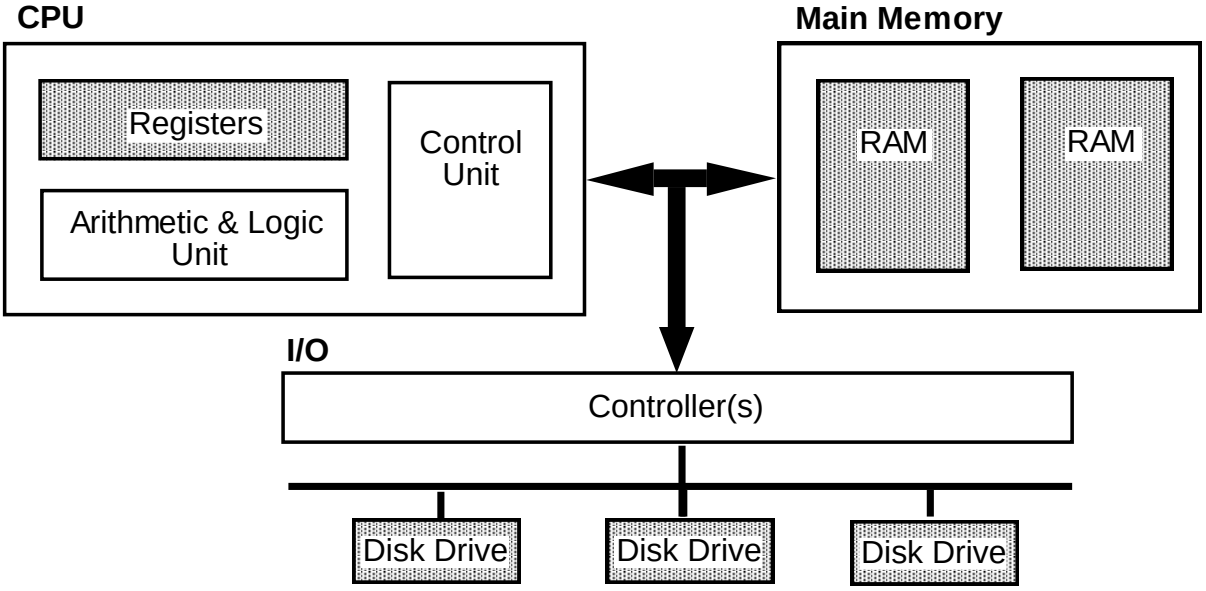
- It supports shorter control words.
- It supports easy implementation of new conrol signals therefore it is more flexible.
- It allows low degree of parallelism i.e., degree of parallelism is either 0 or 1.
- Requires an additional hardware (decoders) to generate control signals, it implies it is slower than horizontal microprogrammed.
- It is less flexible than horizontal but more flexible than that of hardwired control unit.

UNIT III

Memory Organization: Characteristics of Memory Systems:

Main Memory (RAM) Organisation

Computers employ many different types of memory (semi-conductor, magnetic disks and tapes, DVDs etc.) to hold data and programs. Each type has its own characteristics and uses. We will look at the way that Main Memory (RAM) is organised and very briefly at the characteristics of Register Memory and Disk Memory. Let's locate these 3 types of memory in an abstract computer:



Register Memory

Registers are memories located within the Central Processing Unit (CPU). They are few in number (there are rarely more than 64 registers) and also small in size, typically a register is less than 64 bits; 32-bit and more recently 64-bit are common in desktops.

The contents of a register can be "read" or "written" very quickly[1] however, often an order of magnitude faster than main memory and several orders of magnitude faster than disk memory.

Different kinds of register are found within the CPU. General Purpose Registers[2] are available for general[3] use by the programmer. Unless the context implies otherwise we'll use the term "register" to refer to a General Purpose Register within the CPU. Most modern CPU's have between 16 and 64 general purpose registers. Special Purpose Registers have specific uses and are either non-programmable and internal to the CPU or accessed with special instructions by the programmer. Examples of such registers that we will encounter later in the course include: the Program Counter register (PC), the Instruction Register (IR), the ALU Input & Output registers, the Condition Code (Status/Flags) register, the Stack Pointer register (SP). The size (the number of bits in the register) of the these registers varies according to register type. The Word Size of an architecture is often (but not always!) defined by the size of the general purpose registers.

---

1 e.g. less than a nanosecond (10^{-9} sec)

2 Occasionally called Working Registers

3 Used for performing calculations, moving and manipulating data etc.

In contrast to main memory and disk memory, registers are referenced directly by specific instructions or by encoding a register number within a computer instruction. At the programming (assembly) language level of the CPU, registers are normally specified with special identifiers (e.g. R0, R1, R7, SP, PC)

As a final point, the contents of a register are lost if power to the CPU is turned off, so registers are unsuitable for holding long-term information or information that is needed for retention after a power-shutdown or failure. Registers are however, the fastest memories, and if exploited can result in programs that execute very quickly.

Main Memory (RAM)

If we were to sum all the bits of all registers within CPU, the total amount of memory probably would not exceed 5,000 bits. Most computational tasks undertaken by a computer require a lot more memory. Main memory is the next[4] fastest memory within a computer and is much larger in size. Typical main memory capacities for different kinds of computers are: PC 512MB[5], fileserver 2GB, database server 8GB. Computer architectures also impose an architectural constraint on the maximum allowable RAM. This constraint is normally equal to $\mathbf{2^{WordSize}}$ memory locations.

RAM[6] (Random[7] Access Memory) is the most common form of Main Memory. RAM is normally located on the motherboard and so is typically less than 12 inches from the CPU. ROM (Read Only Memory) is like RAM except that its contents cannot be overwritten and its contents are not lost if power is turned off (ROM is non-volatile).

Although slower than register memory, the contents of any location[8] in RAM can still be "read" or "written" very quickly[9]. The time to read or write is referred to as the **access time** and is constant for all RAM locations.

In contrast to register memory, RAM is used to hold both program code (instructions) and data (numbers, strings etc). Programs are "loaded" into RAM from a disk prior to execution by the CPU.

Locations in RAM are identified by an **addressing scheme** *e.g.* numbering the bytes in RAM from 0 onwards[10]. Like registers, the contents of RAM are lost if the power is turned off.

Disk Memory

Disk memory[11] is used to hold programs and data over the longer term. The **contents of a disk are NOT lost if the power is turned off.** Typical hard disk capacities range from 40GB to over 500 GB ($5x10^{29}$). Disks are much slower than register and main memory, the access-time (known as the seek-time) to data on disk is typically between 2 and 4 milli-

---

4 Actually many computers systems also include Cache memory, which is faster than Main memory, but slower than register memory. We will ignore Cache memories in this course.

5 $1K = 2^{10} = 1024$, $1M = 2^{20}$, $1G = 2^{30}$, 'B' will be used for Bytes, and 'b' or 'bit' for bits, cf. 1MB and 1Mbit

6 There are many types of RAM technologies.

7 Random is a Misnomer. Direct Access Memory would have been a better term.

8 Typically a byte multiple.

9 e.g. less than 10 nanoseconds ($10x10^{-9}$ sec)

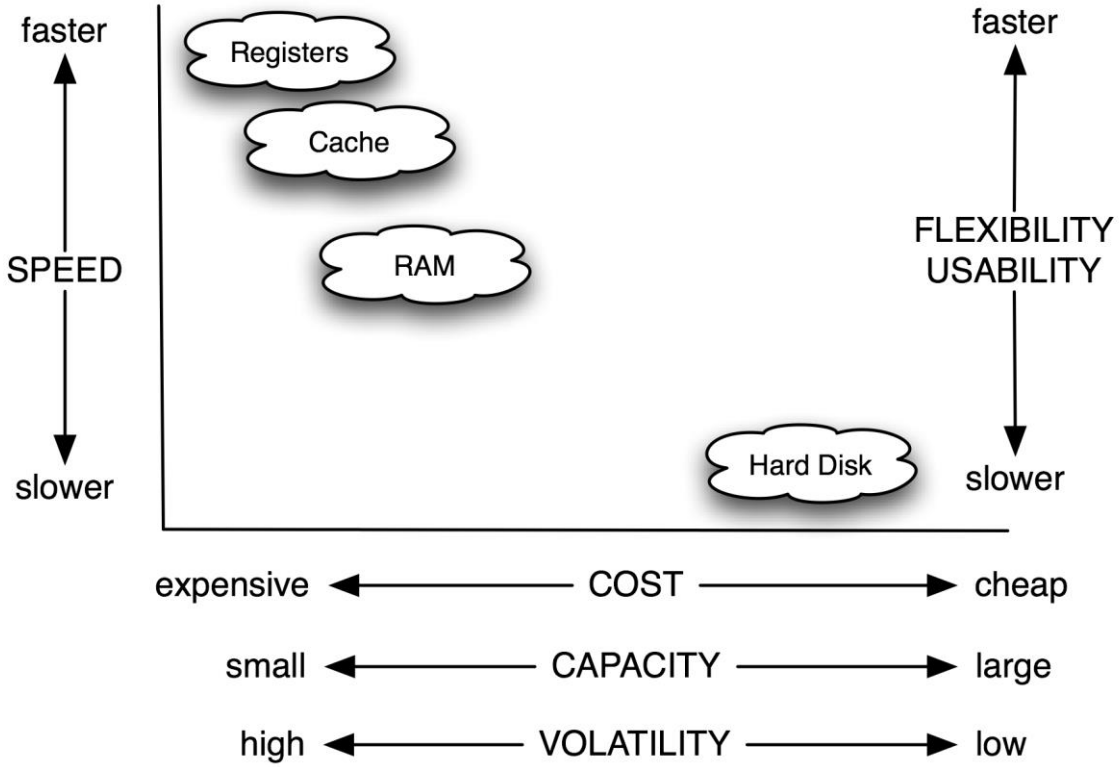
10 Some RAM locations (typically those with the lowest & highest addresses) may cause side-effects, e.g. cause data to be transferred to/from external devices

11 Some authors refer to disk memory as disk storage.

seconds, although disk drives can transfer thousands of bytes in one go achieving transfer rates from 25MB/s to 500MB/s.

Disks can be housed internally within a computer "box" or externally in an enclosure connected by a fast USB or firewire cable[12]. Disk locations are identified by special disk addressing schemes (e.g. track and sector numbers).

Summary of Characteristics



SRAM, DRAM, SDRAM, DDR SDRAM

There are many kinds of RAM and new ones are invented all the time. One of aims is to make RAM access as fast as possible in order to keep up with the increasing speed of CPUs.

SRAM (Static RAM) is the fastest form of RAM but also the most expensive. Due to its cost it is not used as main memory but rather for cache memory. Each bit requires a 6-transistor circuit.

DRAM (Dynamic RAM) is not as fast as SRAM but is cheaper and is used for main memory. Each bit uses a single capacitor and single transistor circuit. Since capacitors lose their charge, DRAM needs to be refreshed every few milliseconds. The memory system does this transparently. There are many implementations of DRAM, two well-known ones are SDRAM and DDR SDRAM.

SDRAM (Synchronous DRAM) is a form of DRAM that is synchronised with the clock of the CPU's system bus, sometimes called the front-side bus (FSB). As an example, if the system bus operates at 167Mhz over an 8-byte (64-bit) data bus , then an SDRAM module could transfer 167 x 8 ~ 1.3GB/sec.

DDR SDRAM (Double-Data Rate DRAM) is an optimisation of SDRAM that allows data to be transferred on both the rising edge and falling edge of a clock signal. Effectively doubling

[12] For details about how disks and other storage devices work, check out Tanenbaum or Stallings.

the amount of data that can be transferred in a period of time. For example a PC-3200 DDR-SDRAM module operating at 200Mhz can transfer 200 x 8 x 2 ~ 3.2GB/sec over an 8-byte (64-bit) data bus.

ROM, PROM, EPROM, EEPROM, Flash

In addition to RAM, they are also a range of other semi-conductor memories that retain their contents when the power supply is switched off.

ROM (Read Only Memory) is a form of semi-conductor that can be written to once, typically in bulk at a factory. ROM was used to store the "boot" or start-up program (so called firmware) that a computer executes when powered on, although it has now fallen out-of-favour to more flexible memories that support occasional writes. ROM is still used in systems with fixed functionalities, e.g. controllers in cars, household appliances etc.

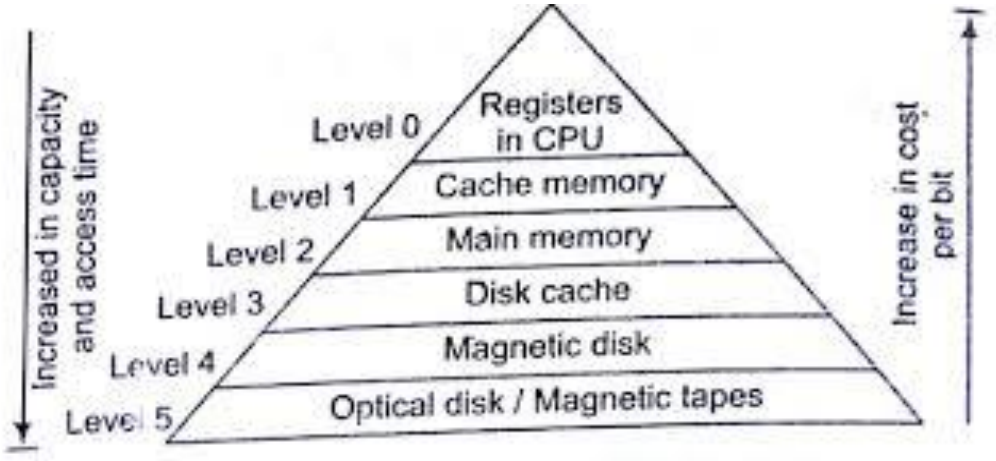
PROM (Programmable ROM) is like ROM but allows end-users to write their own programs and data. It requires a special PROM writing equipment. Note: users can only write-once to PROM.

EPROM (Erasable PROM). With EPROM we can erase (using strong ultra-violet light) the contents of the chip and rewrite it with new contents, typically several thousand times. It is commonly used to store the "boot" program of a computer, known as the firmware. PCs call this firmware, the BIOS (Basic I/O System). Other systems use Open Firmware. Intel-based Macs use EFI (Extensible Firmware Interface).

EEPROM (Electrically Erasable PROM). As the name implies the contents of EEPROMs are erased electrically. EEPROMSs are also limited to the number of erase-writes that can be performed (e.g, 100,000) but support updates (erase-writes) to individual bytes whereas EPROM updates the whole memory and only supports around 10,000 erase-write cycles.

FLASH memory is a cheaper form of EEPROM where updates (erase-writes) can only be performed on blocks of memory, not on individual bytes. Flash memories are found in USB sticks, flash cards and typically range in size from 32M to 2GB. The number of erase/write cycles to a block is typically several hundred thousand before the block can no longer be written.

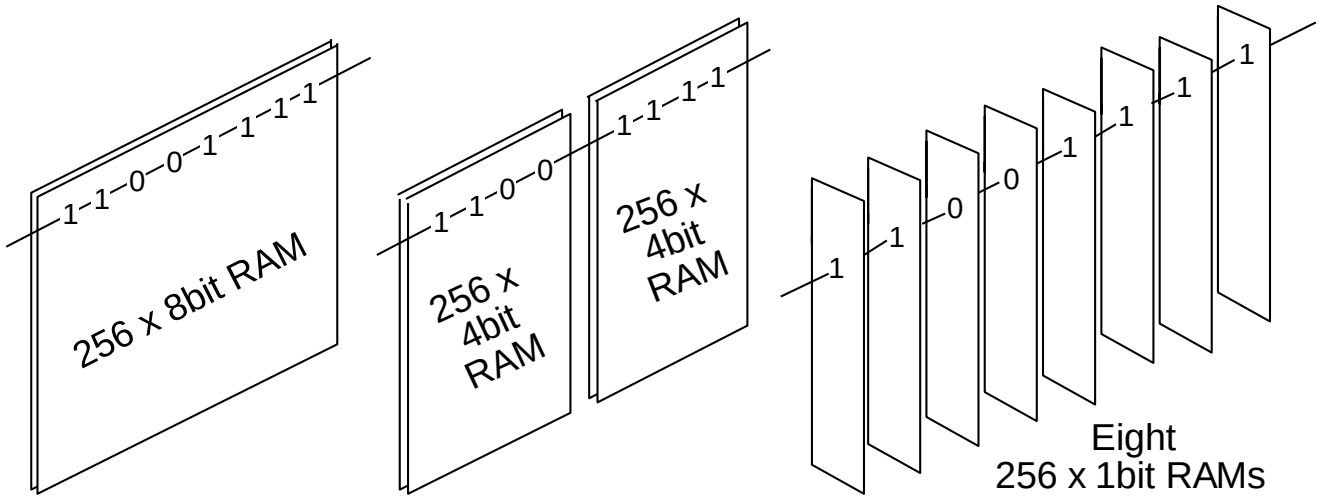
Types of Memory:



Design of memory subsystem using Static, Dynamic Memory chips:

Memory Modules, Memory Chips

So far, we have looked at the logical organisation of main memory. Physically RAM comes on small memory modules (little green printed circuit-boards about the size of a finger). A typical memory module holds 512MB to 2GB. The computer's motherboard will have slots to hold 2, 4 maybe 8 memory modules. Each memory module is itself comprised of several memory chips. For example here are 3 ways of forming a 256x8 bit memory module.

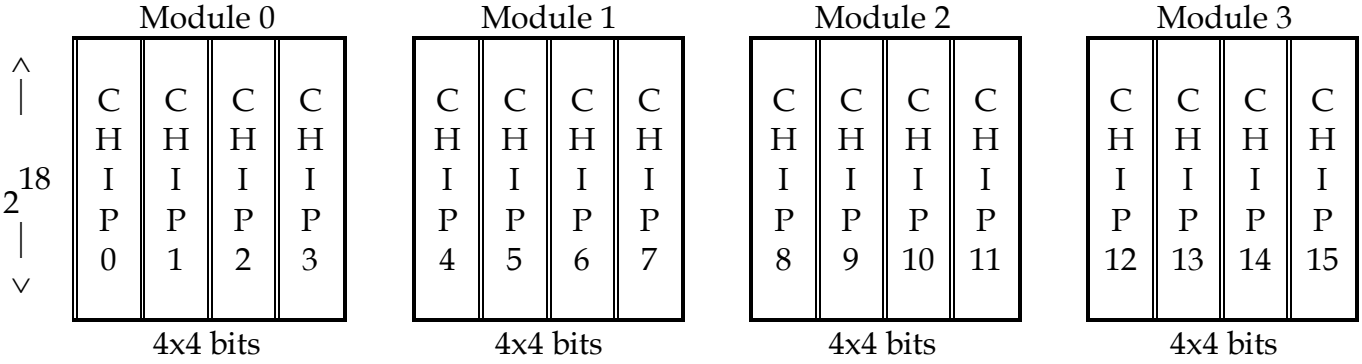


In the first case, main memory is built with a single memory chip. In the second, we use two memory chips, one gives us the most significant 4 bits, the other, the least significant 4 bits. In the third we use 8 memory chips, each chip gives us 1 bit - to read an 8 bit memory word, we would have to access all 8 memory chips simultaneously and concatenate the bits.

On PCs, memory modules are known as DIMMs (dual inline memory modules) and support 64-bit transfers. The previously generation of modules were called SIMMs (single inline memory modules) and supported 32-bit data transfers.

Example: Given Main Memory = 1M x 16 bit (word addressable),

RAM chips = 256K x 4 bit



$$\text{RAM chips per memory module} = \frac{\text{Width of Memory Word}}{\text{Width of RAM Chip}} = 16/4 = 4$$

18 bits are required to address a RAM chip (since 256K = 2^{18} = Length of RAM Chip)

A 1Mx16 bit word-addressed memory requires 20 address bits (since 1M =2^{20})

Therefore 2 bits (=20–18) are needed to select a module.

The total number of RAM Chips = (1M x 16) / (256K x 4) = 16.

Total number of Modules = Total number of RAM chips / RamChipsPerModule = 16/4 = 4

Interleaved Memory

When memory consists of several memory modules, some address bits will select the module, and the remaining bits will select a row within the selected module.

When the module selection bits are the least significant bits of the memory address we call the resulting memory a **low-order interleaved** memory.

When the module selection bits are the most significant bits of the memory address we call the resulting memory a **high-order interleaved** memory.

Interleaved memory can yield performance advantages **if** more than one memory module can be read/written at a time:-

(I) for low-order interleave if we can read the same row in each module. This is good for a single multi-word access of sequential data such as program instructions, or elements in a vector,

(ii) for high-order interleave, if different modules can be independently accessed by different units. This is good if the CPU can access rows in one module, while at the same time, the hard disk (or a second CPU) can access different rows in another module.

Example: Given that Main Memory = 1Mx8bits, RAM chips = 256K x 4bit. For this memory we would require 4x2=8 RAM chips. Each chip would require 18 address bits (ie. 2^{18} = 256K) and the full 1Mx16 bit memory would requires 20 address bits (ie. 2^{20} = 1M)

High Speed Memories:

Cache Memory:

Cache Memory in Computer Organization:

Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.

Elements of Cache Design:

Levels of memory:

- **Level 1 or Register –**
  It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.

- **Level 2 or Cache memory –**
  It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory –**
  It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory –**
  It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

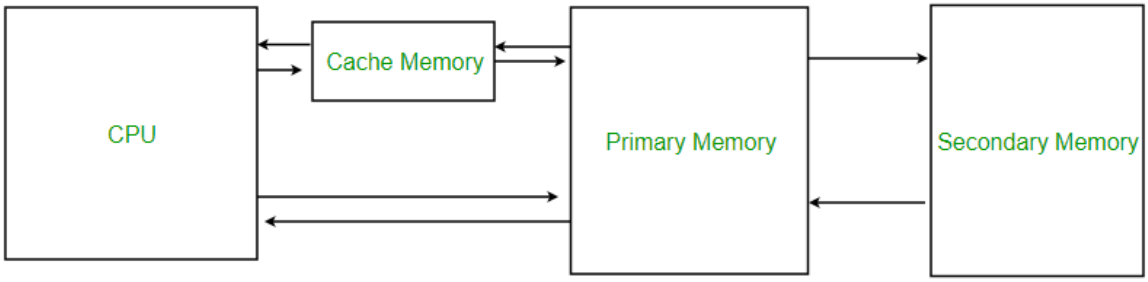
- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

```
Hit ratio = hit / (hit + miss) =  no. of hits/total accesses
```

We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Structure of cache and main memory:



Mapping functions:

Cache Mapping:

There are three different types of mapping used for the purpose of cache memory which are as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained below.

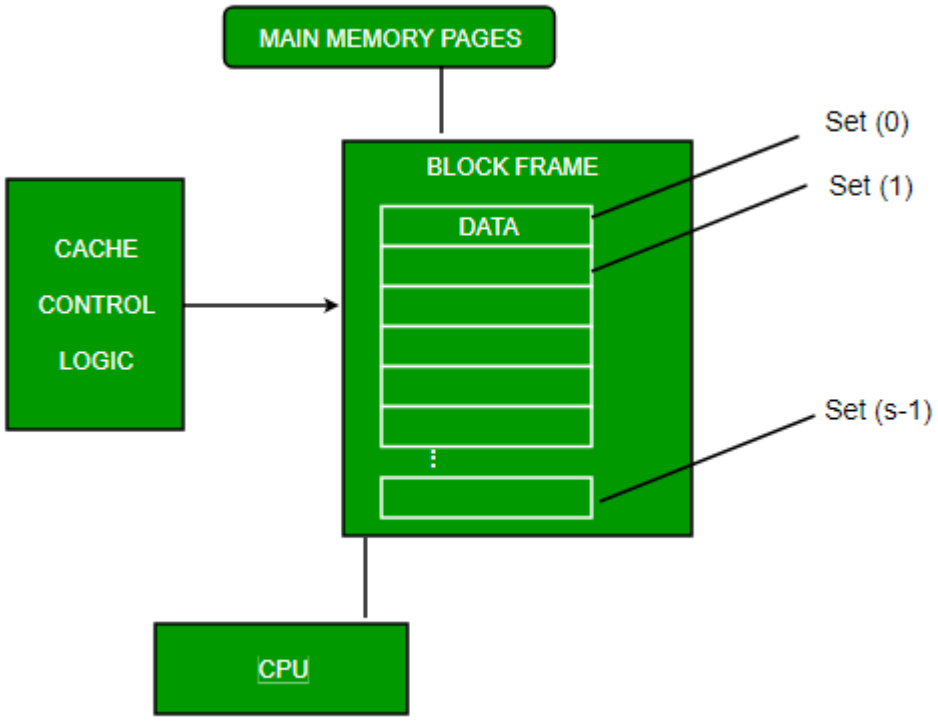
1. **Direct Mapping –**
   The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or
   In Direct mapping, assigne each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in

the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

2. `i = j modulo m`
3. `where`
4. `i=cache line number`
5. `j= main memory block number`

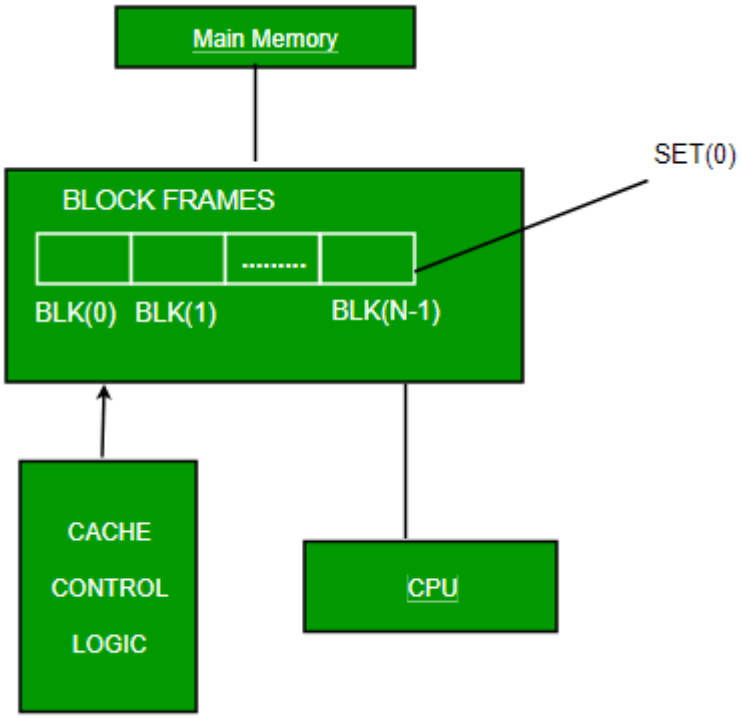
   `m=number of lines in the cache`

   For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks of main memory. The cache logic interprets these s bits as a tag of s-r bits (most significant portion) and a line field of r bits. This latter field identifies one of the $m=2^r$ lines of the cache.



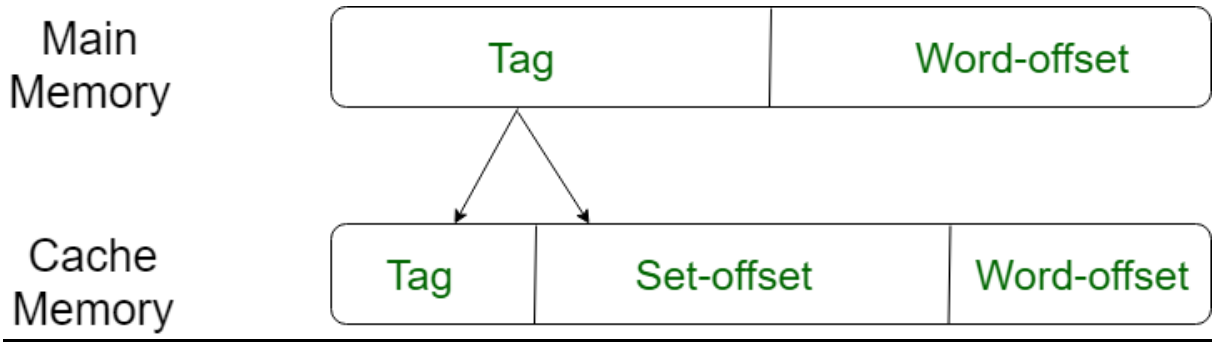
Associative Mapping –

In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.



Set-associative Mapping –
This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***. Then a block in memory can map to any one of the lines of a specific set..Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques.
In this case, the cache consists of a number of sets, each of which consists of a number of lines. The relationships are

```
m = v * k

i= j mod v


where

i=cache set number

j=main memory block number

v=number of sets

m=number of lines in the cache number of sets

k=number of lines in each set
```



Application of Cache Memory –

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

Types of Cache –

- **Primary Cache –**
  A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
- **Secondary Cache –**
  Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.

Replacement algorithms:

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

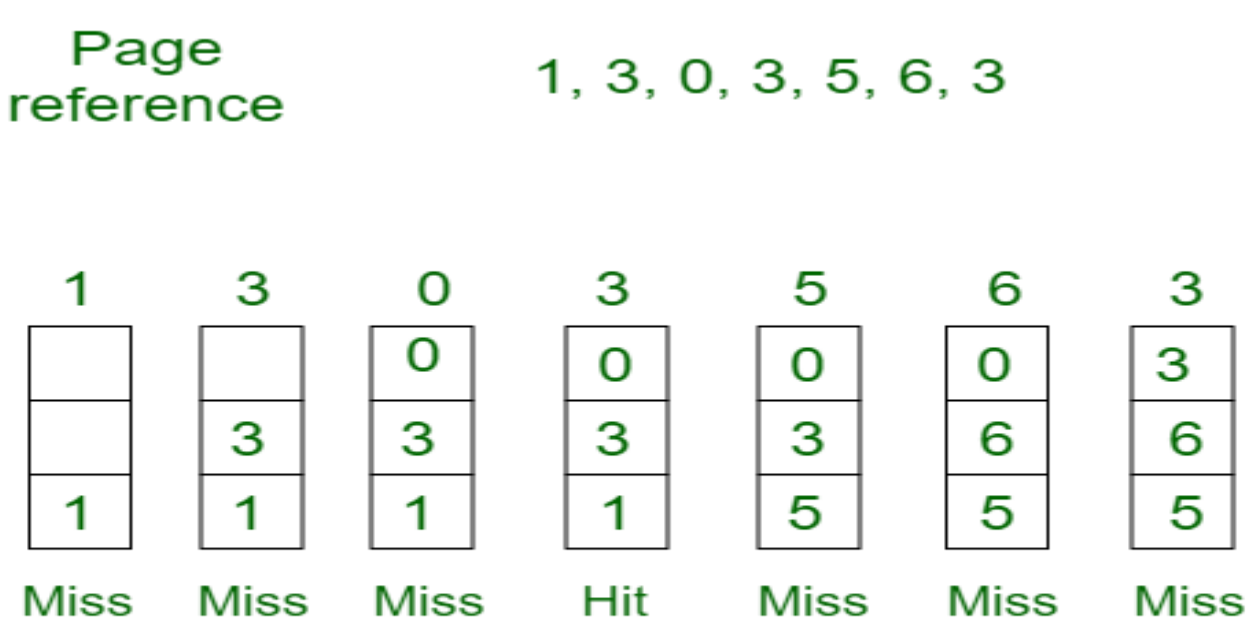
Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.
Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms:

First In First Out (FIFO)

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example-1 Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames.Find number of page faults.

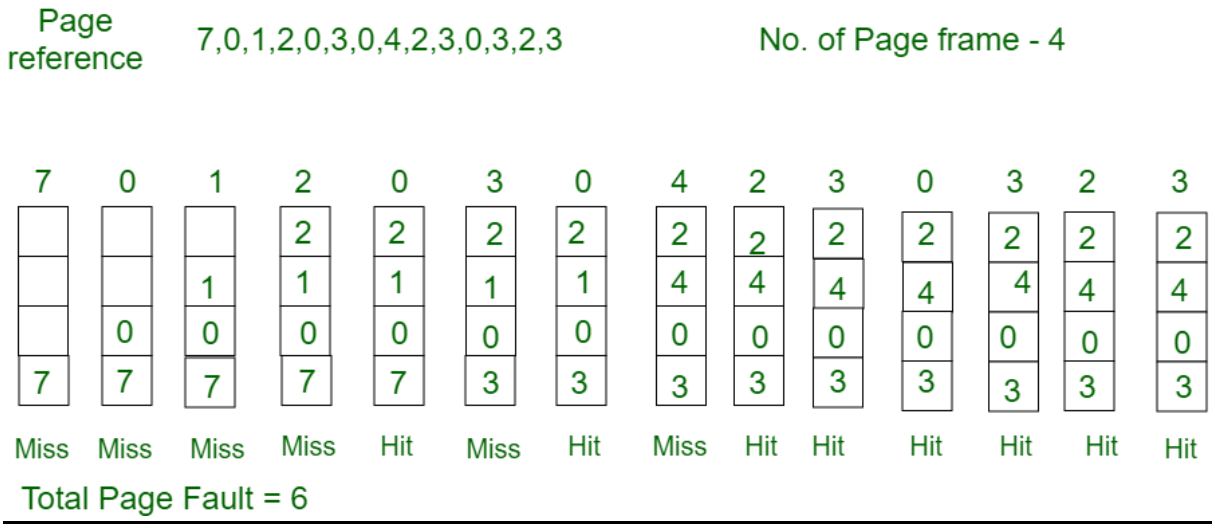


- Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**
  when 3 comes, it is already in memory so —> **0 Page Faults.**
  Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**
  6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**

- Finally when 3 come it is not avilable so it replaces 0 **1 page fault**
  Belady's anomaly **–** Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

2.Optimal Page replacement :

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.

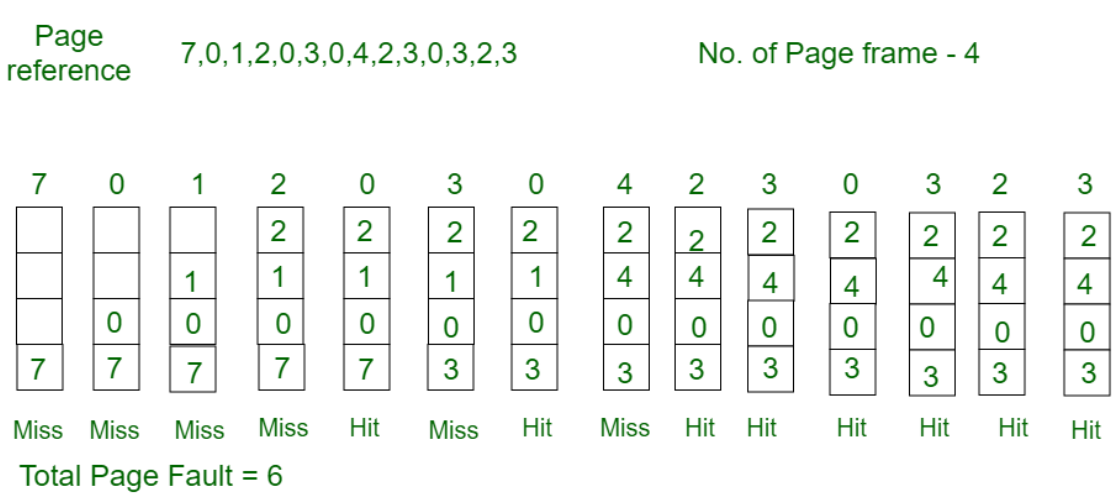


- Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
  0 is already there so —> **0 Page fault.**
  when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
  0 is already there so —> **0 Page fault.**.
  4 will takes place of 1 —> **1 Page Fault.**

  Now for the further page reference string —> **0 Page fault** because they are already available in the memory.
  Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

3.Least Recently Used :

In this algorithm page will be replaced which is least recently used.
Example-3 Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames.Find number of page faults.



Here LRU has same number of page fault as optimal but it may differ according to question.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already their so —> **0 Page fault.**

when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
0 is already in memory so —> **0 Page fault**.
4 will takes place of 1 —> **1 Page Fault**
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

External Memory:

Types of External Memory

Magnetic Disk

RAID Removable Optical

CD-ROM CD-Writable

(WORM) CD-R/W DVD

Magnetic Tape

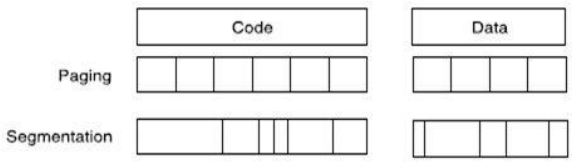
Virtual memory:

Physical and **virtual memory** are forms of **memory** (internal storage of data). **Physical memory** exists on chips (**RAM memory**) and on storage devices such as hard disks. ... **Virtual memory** is a process whereby data (e.g., programming code,) can be rapidly exchanged **between physical memory** storage locations and **RAM memory**.

Virtual memory:

Why Virtual memory?

- Allows applications to be bigger than main memory size
- Helps with multiple process management
  - Each process gets its own chunk of memory
  - Protection of processes against each other
  - Mapping of multiple processes to memory
  - Relocation
  - Application and CPU run in virtual space
  - Mapping of virtual to physical space is invisible to the application
- Management between main memory and disk
  - Miss in main memory is a page fault or address fault
  - Block is a page or segment



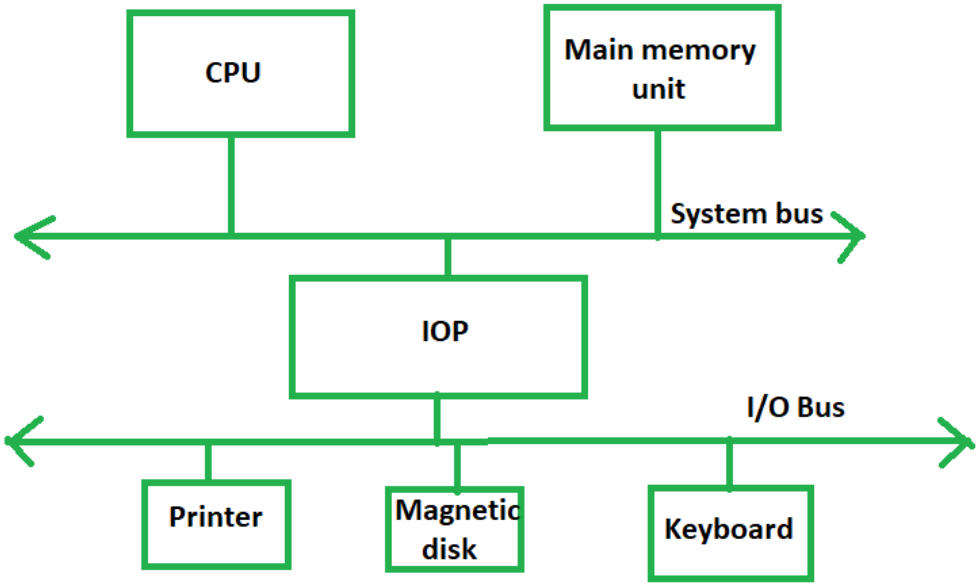
UNIT IV

I/O Organization:

Introduction of Input-Output Processor:

The **DMA mode** of data transfer reduces CPU's overhead in handling I/O operations. It also allows parallelism in CPU and I/O operations. Such parallelism is necessary to avoid wastage of valuable CPU time while handling I/O devices whose speeds are much slower as compared to CPU. The concept of DMA operation can be extended to relieve the CPU further from getting involved with the execution of I/O operations. This gives rises to the development of special purpose processor called **Input-Output Processor (IOP) or IO channel**.
The Input Output Processor (IOP) is just like a CPU that handles the details of I/O operations. It is more equipped with facilities than those are available in typical DMA controller. The IOP can fetch and execute its own instructions that are specifically designed to characterize I/O transfers. In addition to the I/O – related tasks, it can perform other processing tasks like arithmetic, logic, branching and code translation. The main memory unit takes the pivotal role. It communicates with processor by the means of DMA.

The block diagram –



Input-Output Processor

The Input Output Processor is a specialized processor which loads and stores data into memory along with the execution of I/O instructions. It acts as an interface between system and devices. It involves a sequence of events to executing I/O operations and then store the results into the memory.

Advantages –

- The I/O devices can directly access the main memory without the intervention by the processor in I/O processor based systems.
- It is used to address the problems that are arises in Direct memory access method.

Input / Output Module:

Need, Techniques Interrupt Driven I/O:

Interrupt driven I/O is an alternative scheme dealing with I/O. Interrupt I/O is a way of controlling input/output activity whereby a peripheral or terminal that needs to make or receive a data transfer sends a signal. This will cause a program interrupt to be set. At a time appropriate to the priority level of the I/O interrupt. Relative to the total interrupt system, the processors enter an interrupt service routine. The function of the routine will depend upon the system of interrupt levels and priorities that is implemented in the processor. The interrupt technique requires more complex hardware and software, but makes far more efficient use of the computer's time and capacities. Figure 2 shows the simple interrupt processing.

For **input**, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports or memory mapping.

For **output**, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.
Here the CPU works on its given tasks continuously. When an input is available, such as when someone types a key on the keyboard, then the CPU is interrupted from its work to take care of the input data. The CPU can work continuously on a task without checking the input devices, allowing the devices themselves to interrupt it as necessary.

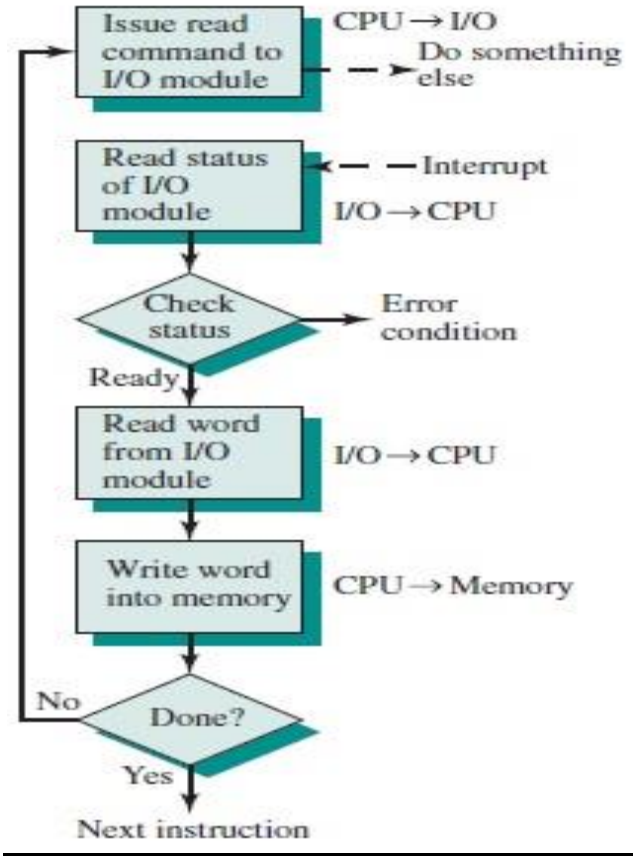
Basic Operations of Interrupt

1. CPU issues read command.
2. I/O module gets data from peripheral whilst CPU does other work.
3. I/O module interrupts CPU.
4. CPU requests data.
5. I/O module transfers data.

Interrupt Processing

1. A device driver initiates an I/O request on behalf of a process.
2. The device driver signals the I/O controller for the proper device, which initiates the requested I/O.
3. The device signals the I/O controller that is ready to retrieve input, the output is complete or that an generated.

4. The CPU receives the interrupt signal on the interrupt-request line and transfer control over the i routine.
5. The interrupt handler determines the cause of the interrupt, performs the necessary processing and ex *from*" interrupt instruction.
6. The CPU returns to the execution state prior to the interrupt being signaled.
7. The CPU continues processing until the cycle begins again.



Basic concepts of an Interrupt :

Interrupt is the mechanism by which modules like I/O or memory may interrupt the normal processing by CPU. It may be either clicking a mouse, dragging a cursor, printing a document etc the case where interrupt is getting generated.

Why we require Interrupt?

External devices are comparatively slower than CPU. So if there is no interrupt CPU would waste a lot of time waiting for external devices to match its speed with that of CPU. This decreases the efficiency of CPU. Hence, interrupt is required to eliminate these limitations.

With Interrupt:

Response of CPU to an Interrupt:

1. Suppose CPU instructs printer to print a certain document.
2. While printer does its task, CPU engaged in executing other tasks.
3. When printer is done with its given work, it tells CPU that it has done with its work. (The word 'tells' here is interrupt which sends one message that printer has done its work successfully.).

Advantages:

- It increases the efficiency of CPU.
- It decreases the waiting time of CPU.
- Stops the wastage of instruction cycle.

Disadvantages:

- CPU has to do a lot of work to handle interrupts, resume its previous execution of programs (in short, overhead required to handle the interrupt request.).

Design Issues:

Design Issues

There are 2 main problems for interrupt I/O, which are:

- There are multiple I/O modules, how should the processor determine the device that issued the interrupt signal?
- How does the processor decide which module to process when multiple interrupts have occurred?

There are 4 main ways to counter these problems, which are:

- Multiple Interrupt Lines
- Software Poll
- Daisy Chain (Hardware Poll, Vectored)
- Bus Arbitration (Vectored)

Difference between Maskable and Non Maskable Interrupt

An interrupt is an event caused by a component other than the CPU. It indicates the CPU of an external event that requires immediate attention. Interrupts occur asynchronously. Maskable and non-maskable interrupts are two types of interrupts.

1. Maskable Interrupt :

An Interrupt that can be disabled or ignored by the instructions of CPU are called as Maskable Interrupt.The interrupts are either edge-triggered or level-triggered or level-triggered.

Eg:

```
RST6.5,RST7.5,RST5.5 of 8085
```

2. Non-Maskable Interrupt :

An interrupt that cannot be disabled or ignored by the instructions of CPU are called as Non-Maskable Interrupt.A Non-maskable interrupt is often used when response time is critical or when an interrupt should never be disable during normal system operation. Such uses include reporting non-recoverable hardware errors, system debugging and profiling and handling of species cases like system resets.

Eg:

```
Trap of 8085
```

Difference between maskable and nonmaskable interrupt :

SR.NO.	Maskable Interrupt	Non Maskable Interrupt
1	Maskable interrupt is a hardware Interrupt that can be disabled or ignored by the instructions of CPU.	A non-maskable interrupt is a hardware interrupt that cannot be disabled or ignored by the instructions of CPU.
2	When maskable interrupt	When non-maskable interrupts

SR.NO.	Maskable Interrupt	Non Maskable Interrupt
	occur, it can be handled after executing the current instruction.	occur, the current instructions and status are stored in stack for the CPU to handle the interrupt.
3	Maskable interrupts help to handle lower priority tasks.	Non-maskable interrupt help to handle higher priority tasks such as watchdog timer.
4	Maskable interrupts used to interface with peripheral device.	Non maskable interrupt used for emergency purpose e.g power failure, smoke detector etc .
5	In maskable interrupts, response time is high.	In non maskable interrupts, response time is low.
6	It may be vectored or non-vectored.	All are vectored interrupts.
7	Operation can be masked or made pending.	Operation Cannot be masked or made pending.
8	RST6.5, RST7.5, and RST5.5 of 8085 are some common examples of maskable Interrupts.	Trap of 8085 microprocessor is an example for non-maskable interrupt.

Priorities, Interrupt handling:

Priority Interrupts | (S/W Polling and Daisy Chaining)

In I/O Interface (Interrupt and DMA Mode), we have discussed concept behind the Interrupt-initiated I/O. To summarize, when I/O devices are ready for I/O transfer, they generate an interrupt request signal to the computer. The CPU receives this signal, suspends the current instructions it is executing and then moves forward to service that transfer request. But what if multiple devices generate interrupts simultaneously. In that case, we have to have a way to decide which interrupt is to be serviced first. In other words, we have to set a priority among all the devices for systemic interrupt servicing.

The concept of defining the priority among devices so as to know which one is to be serviced first in case of simultaneous requests is called priority interrupt system. This could be done with either software or hardware methods.

SOFTWARE METHOD – POLLING

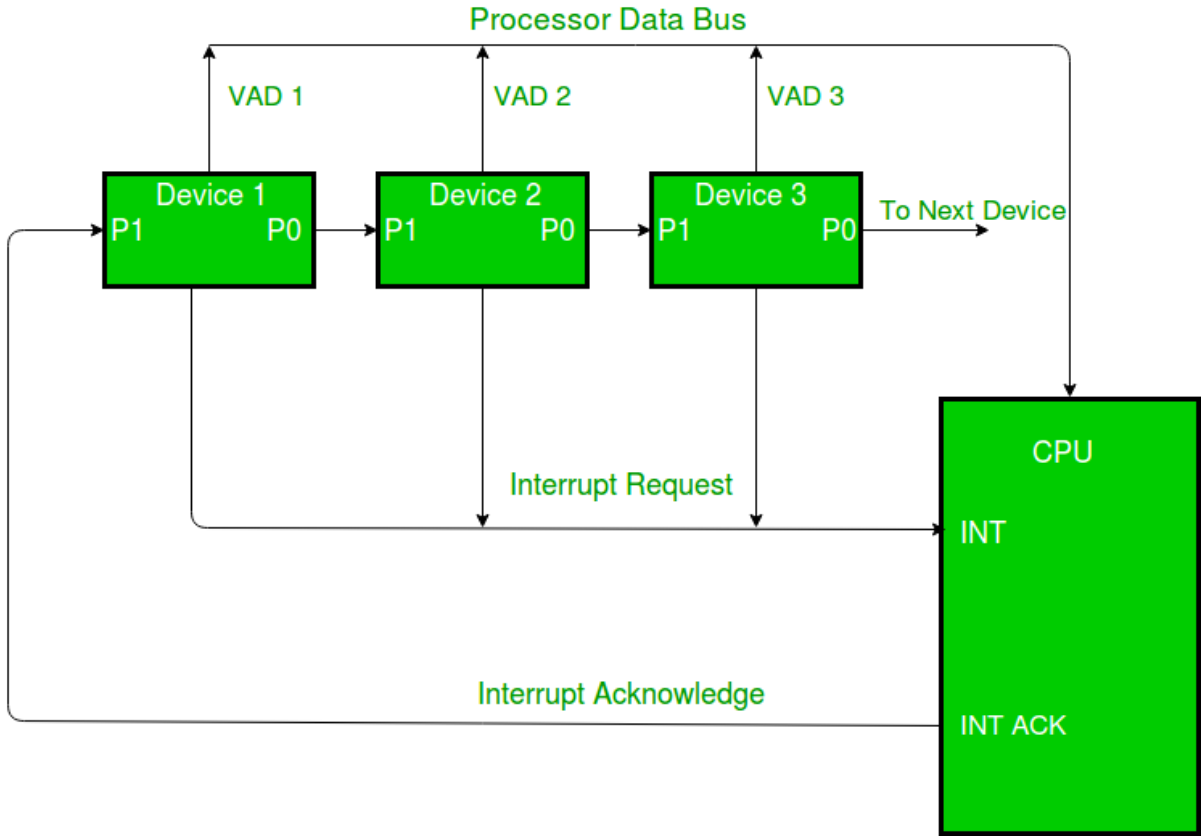
In this method, all interrupts are serviced by branching to the same service program. This program then checks with each device if it is the one generating the interrupt. The order of checking is determined by the priority that has to be set. The device having the highest priority is checked first and then devices are checked in descending order of priority. If the device is checked to be generating the interrupt, another service program is called which works specifically for that particular device.
The structure will look something like this-

```
if (device[0].flag)
    device[0].service();
else if (device[1].flag)
    device[1].service();
.
.
else
    //raise error
```

The major disadvantage of this method is that it is quite slow. To overcome this, we can use hardware solution, one of which involves connecting the devices in series. This is called Daisy-chaining method.

HARDWARE METHOD – DAISY CHAINING

The daisy-chaining method involves connecting all the devices that can request an interrupt in a serial manner. This configuration is governed by the priority of the devices. The device with the highest priority is placed first followed by the second highest priority device and so on. The given figure depicts this arrangement.



WORKING:

There is an interrupt request line which is common to all the devices and goes into the CPU.

- When no interrupts are pending, the line is in HIGH state. But if any of the devices raises an interrupt, it places the interrupt request line in the LOW state.
- The CPU acknowledges this interrupt request from the line and then enables the interrupt acknowledge line in response to the request.

- This signal is received at the PI(Priority in) input of device 1.
- If the device has not requested the interrupt, it passes this signal to the next device through its PO(priority out) output. (PI = 1 & PO = 1)
- However, if the device had requested the interrupt, (PI =1 & PO = 0)
    - The device consumes the acknowledge signal and block its further use by placing 0 at its PO(priority out) output.
    - The device then proceeds to place its interrupt vector address(VAD) into the data bus of CPU.
    - The device puts its interrupt request signal in HIGH state to indicate its interrupt has been taken care of.

  NOTE: VAD is the address of the service routine which services that device.
- If a device gets 0 at its PI input, it generates 0 at the PO output to tell other devices that acknowledge signal has been blocked. (PI = 0 & PO = 0)

Hence, the device having PI = 1 and PO = 0 is the highest priority device that is requesting an interrupt. Therefore, by daisy chain arrangement we have ensured that the highest priority interrupt gets serviced first and have established a hierarchy. The farther a device is from the first device, the lower its priority.

Types Interrupt:

Hardware and Software

1. Hardware Interrupt :
Hardware Interrupt is caused by some hardware device such as request to start an I/O, a hardware failure or something similar. Hardware interrupts were introduced as a way to avoid wasting the processor's valuable time in polling loops, waiting for external events.
For example, when an I/O operation is completed such as reading some data into the computer from a tape drive.

2. Software Interrupt :
Software Interrupt is invoked by the use of INT instruction. This event immediately stops execution of the program and passes execution over to the INT handler. The INT handler is usually a part of the operating system and determines the action to be taken. It occurs when an application program terminates or requests certain services from the operating system.
For example, output to the screen, execute file etc.

Data Transfer Techniques:

Synchronous and Asynchronous Data Transfer:

Computer Organization | Asynchronous input output synchronization:

Asynchronous input output is a form of input output processing that allows others devices to do processing before the transmission or data transfer is done.

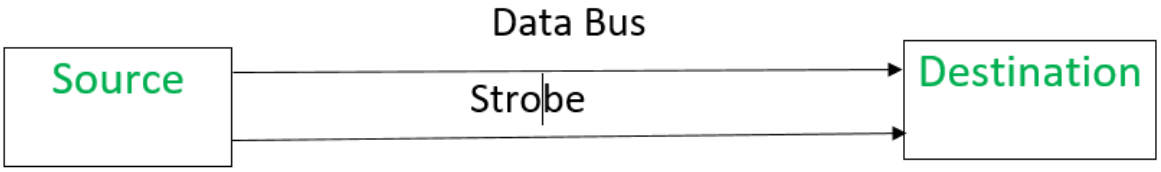
Problem faced in asynchronous input output synchronization –
It is not sure that the data on the data bus is fresh or not as their no time slot for sending or receiving data. This problem is solved by following mechanism:

1. Strobe
2. Handshaking

Data is transferred from source to destination through data bus in between.

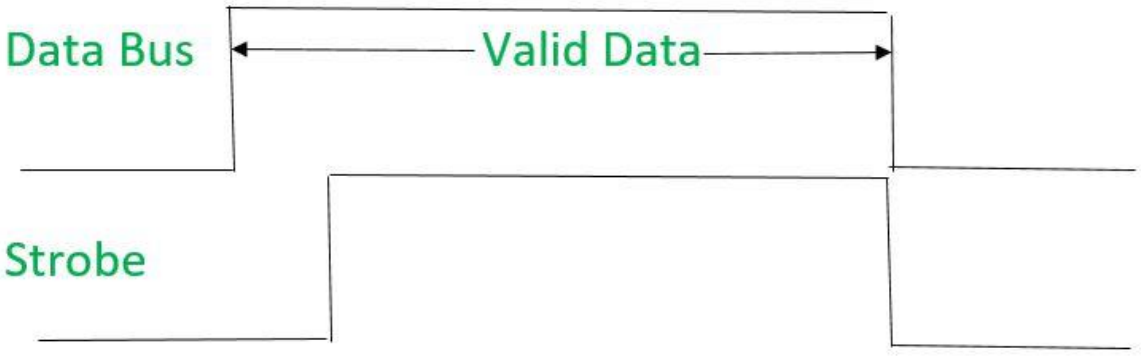
1. Strobe Mechanism:

1. **Source initiated Strobe –** When source initiates the process of data transfer. Strobe is just a signal.



(i) First, source puts data on the data bus and ON the strobe signal.
(ii) Destination on seeing the ON signal of strobe, read data from the data bus.
(iii) After reading data from the data bus by destination, strobe gets OFF.

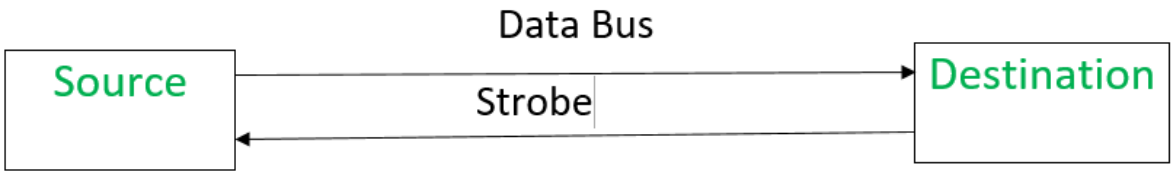
Signals can be seen as:



(i) First, source puts data on the data bus and ON the strobe signal.
(ii) Destination on seeing the ON signal of strobe, read data from the data bus.
(iii) After reading data from the data bus by destination, strobe gets OFF.

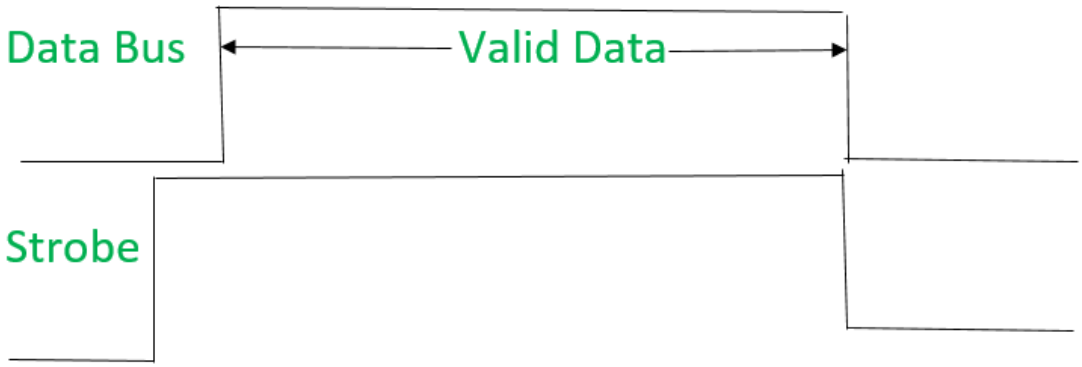
Signals can be seen as:

1. It shows that first data is put on the data bus and then strobe signal gets active.
2. **Destination initiated signal –** When destination initiates the process of data transfer.



i) First, the destination ON the strobe signal to ensure the source to put the fresh data on the data bus.
(ii) Source on seeing the ON signal puts fresh data on the data bus.
(iii) Destination reads the data from the data bus and strobe gets OFF signal.

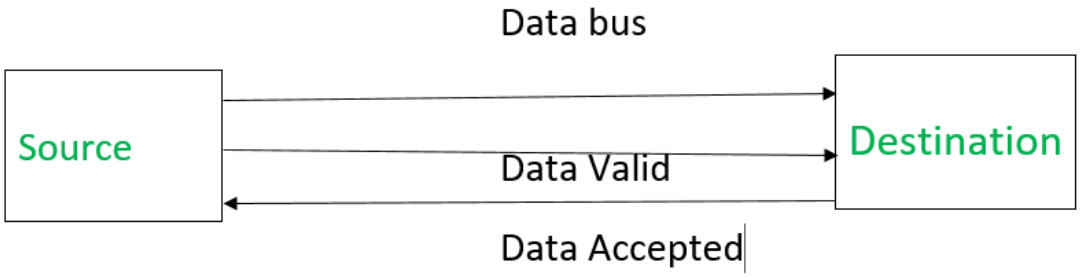
Signals can be seen as:



---

2. Handshaking Mechanism:

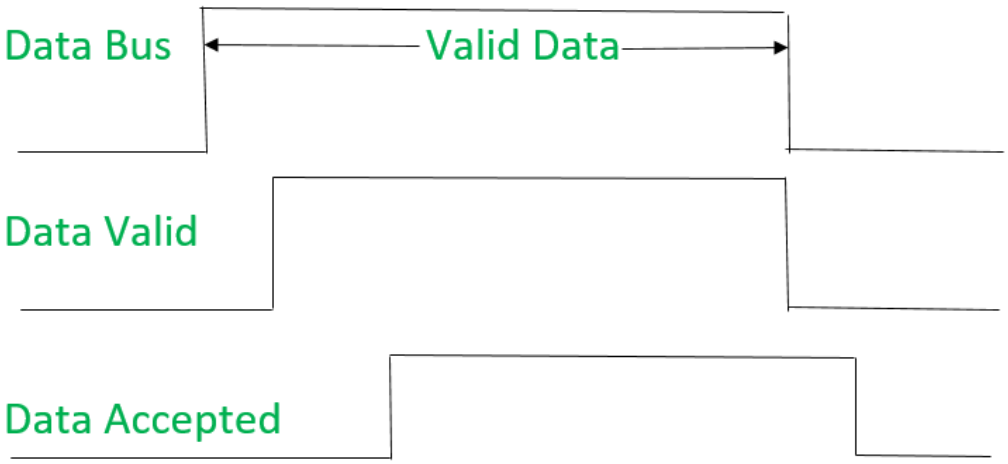
1. **Source initiated Handshaking –** When source initiates the data transfer process. It consists of signals:
   DATA VALID: if ON tells data on the data bus is valid otherwise invalid.
   DATA ACCEPTED: if ON tells data is accepted otherwise not accepted.



(i) Source places data on the data bus and enable Data valid signal.
(ii) Destination accepts data from the data bus and enable Data accepted signal.
(iii) After this, disable Data valid signal means data on data bus is invalid now.
(iv) Disable Data accepted signal and the process ends.

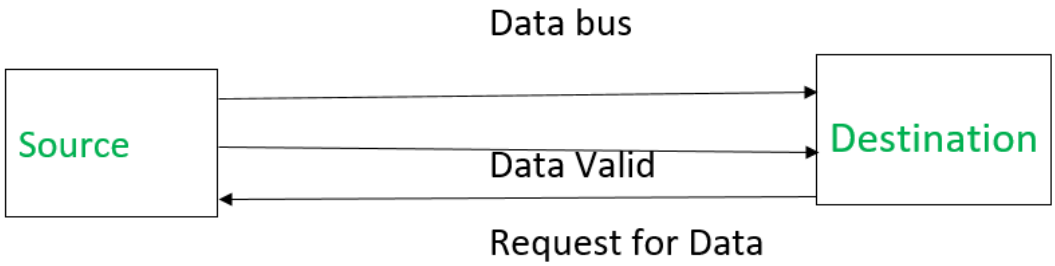
Now there is surety that destination has read the data from the data bus through data accepted signal.

Signals can be seen as:



---

1. It shows that first data is put on the data bus then data valid signal gets active and then data accepted signal gets active. After accepting the data, first data valid signal gets off then data accepted signal gets off.
2. **Destination initiated Handshaking –** When destination initiates the process of data transfer.
   REQUEST FOR DATA: if ON requests for putting data on the data bus.
   DATA VALID: if ON tells data is valid on the data bus otherwise invalid data.



Data Memory Access:

1. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

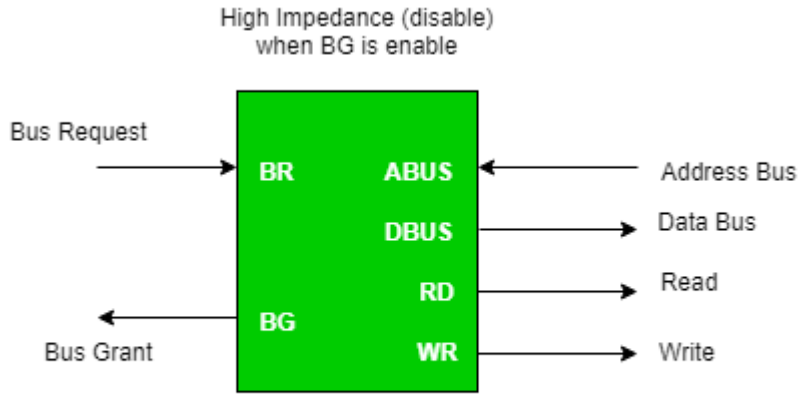


Figure - CPU Bus Signals for DMA Transfer

Bus Request : It is used by the DMA controller to request the CPU to relinquish the control of the buses.
Bus Grant : It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.
Types of DMA transfer using DMA controller:
Burst Transfer :
DMA returns the bus after complete data transfer. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.
Steps involved are:

1. Bus grant request time.
2. Transfer the entire block of data at transfer rate of device because the device is usually slow than the speed at which the data can be transferred to CPU.
3. Release the control of the bus back to CPU
   So, total time taken to transfer the N bytes
   = Bus grant request time + (N) * (memory transfer rate) + Bus release control time.

I/O Interface :

The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.
2. Interrupt- initiated I/O.
3. Direct memory access(DMA).

Now let's discuss each mode one by one.

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is

from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

Note: Both the methods programmed I/O and Interrupt-driven I/O require the active intervention of the
processor to transfer data between memory and the I/O module, and any data transfer must transverse
a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.

3. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

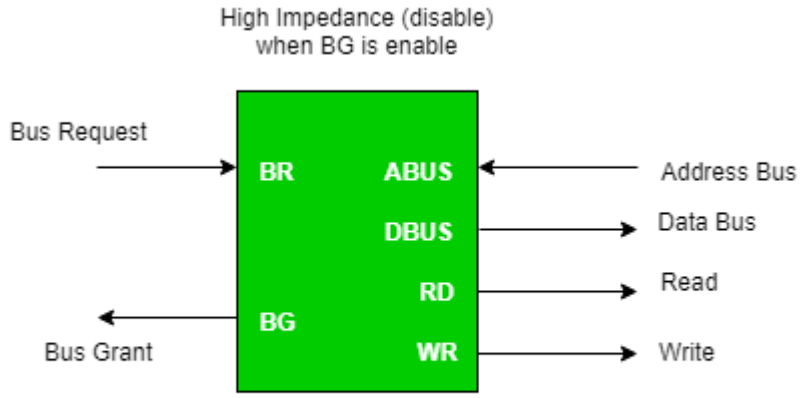
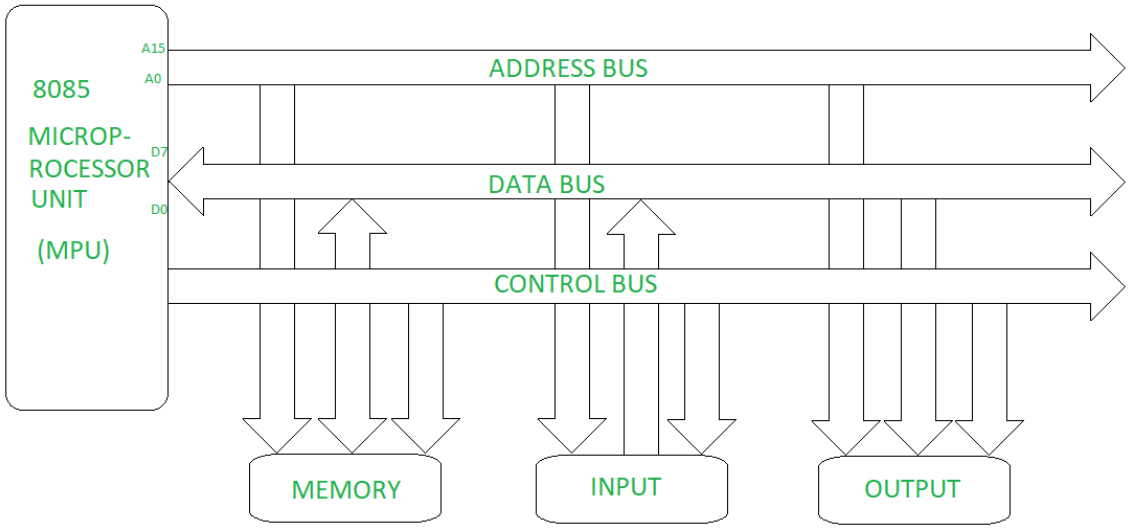


Figure - CPU Bus Signals for DMA Transfer

Buses, Types of buses,

Bus is a group of conducting wires which carries information, all the peripherals are connected to microprocessor through Bus.

Diagram to represent bus organization system of 8085 Microprocessor.



Bus organization system of 8085 Microprocessor

There are three types of buses.

1. **Address bus –**
   It is a group of conducting wires which carries address only.Address bus is unidirectional because data flow in one direction, from microprocessor to memory or from microprocessor to Input/output devices (That is, Out of Microprocessor).

   Length of Address Bus of 8085 microprocessor is 16 Bit (That is, Four Hexadecimal Digits), ranging from 0000 H to FFFF H, (H denotes Hexadecimal). The microprocessor 8085 can transfer maximum 16 bit address which means it can address 65, 536 different memory location.

   The Length of the address bus determines the amount of memory a system can address.Such as a system with a 32-bit address bus can address 2^32 memory locations.If each memory location holds one byte, the addressable memory space is 4 GB.However, the actual amount of memory that can be accessed is usually much less than this theoretical limit due to chipset and motherboard limitations.

2. **Data bus –**
   It is a group of conducting wires which carries Data only.Data bus is bidirectional because data flow in both directions, from microprocessor to memory or Input/Output devices and from memory or Input/Output devices to microprocessor.
   Length of Data Bus of 8085 microprocessor is 8 Bit (That is, two Hexadecimal Digits), ranging from 00 H to FF H. (H denotes Hexadecimal).

   When it is write operation, the processor will put the data (to be written) on the data bus, when it is read operation, the memory controller will get the data from specific memory block and put it into the data bus.

   The width of the data bus is directly related to the largest number that the bus can carry, such as an 8 bit bus can represent 2 to the power of 8 unique values, this equates to the number 0 to 255.A 16 bit bus can carry 0 to 65535.

3. **Control bus –**
   It is a group of conducting wires, which is used to generate timing and control signals to control all the associated peripherals, microprocessor uses control bus to process data, that is what to do with selected memory location. Some control signals are:
   - Memory read
   - Memory write
   - I/O read

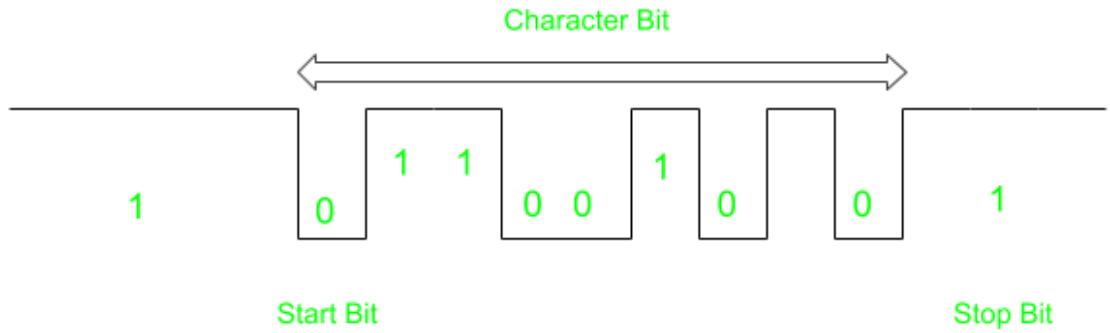
- I/O Write
- Opcode fetch

If one line of control bus may be the read/write line.If the wire is low (no electricity flowing) then the memory is read, if the wire is high (electricity is flowing) then the memory is written.

Serial and parallel transmission:

In most computer **asynchronous mode** of data transfer is used in which two component have a different clock. Data transfer can occur between data in two ways serial and parallel. In case of parallel multiple lines are used to send a single bit whereas in serial transfer each bit is send one at a time. To tell other devices when the character/data will be given a concept of start and end bit is used. A start bit is denoted by 0 and stop bit is detected when line return to 1-state at least one time, here 1-state means that there is not data transfer is occurring.
When a character is not being sent then line is kept in state 1. Start of character is detected when a 0 is sent. The character bit always come after 0 bit. After last bit is sent the state of line to become 1.

The diagram below shows this concept:



Here earlier state of line was 1 when a character has to be send a 0 is send and character bit are transferred.

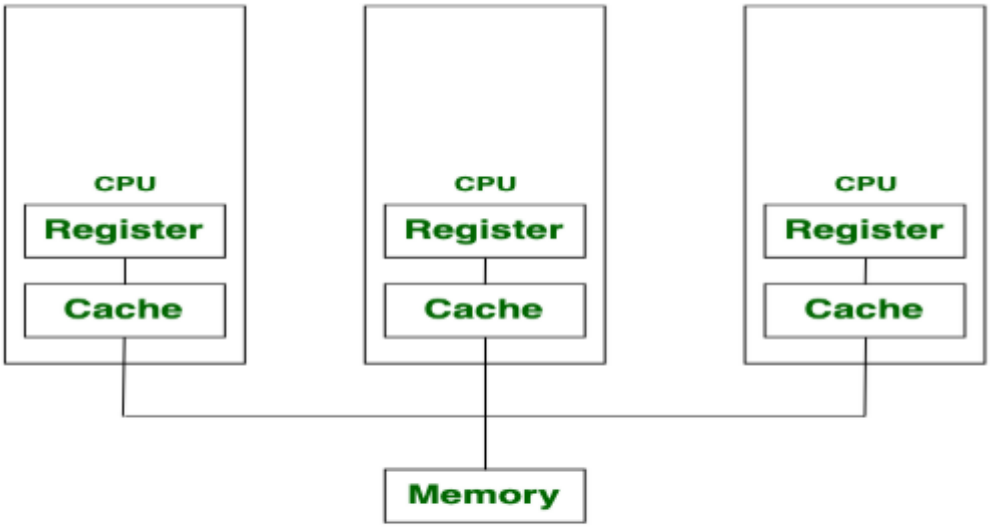
Difference between serial and parallel transfer –

Serial	Parallel
Require single line to send data	Require multiple line
Less error and simple model	Error prone and complex working
Economical	Expensive
Slower data transfer	Faster data transfer
Used for long distance	used for short distance
Example:Computer to Computer	Example:Computer to Printer

I/O, Multiprogramming vs. Multiprocessing :

1. **Multiprocessing :**
   Multiprocessing is a system that has two or more than one processors. In this, CPUs are added for increasing computing speed of the system. Because of

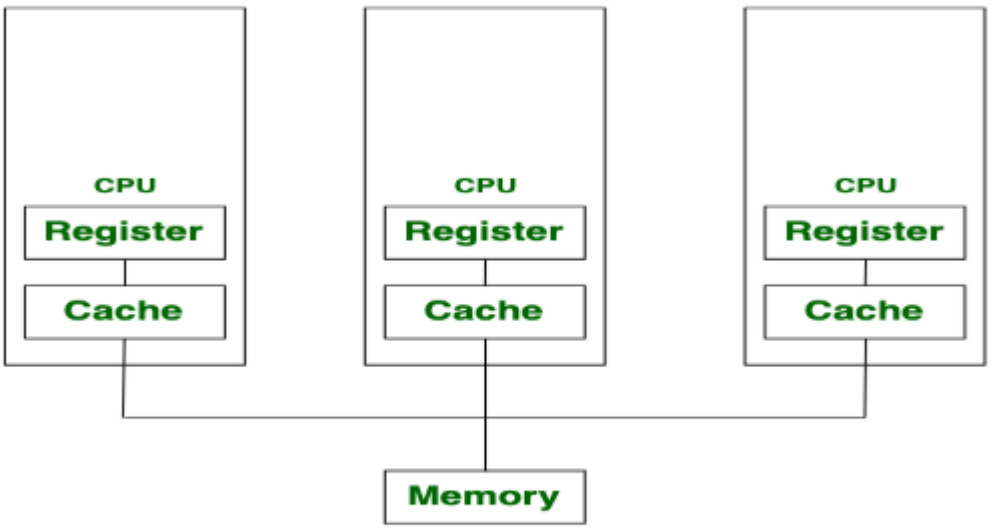
Multiprocessing, there are many processes that are executed simultaneously. Multiprocessing are further classified into two categories: Symmetric Multiprocessing, Asymmetric Multiprocessing.



Multiprocessing

---

2. **Multi-programming :**
Multi-programming is more than one process running at a time, it increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute. The motive is to keep multiple jobs in main memory. If one job gets occupied with Input/output, CPU can be assigned to other job.



Multiprocessing

UNIT V

Microprogramming: Basic Principles:
Features :

Basis Concepts

- Micro-operations: We have already seen that the programs are executed as a sequence of instructions, each instruction consists of a series of steps that make up the instruction cycle fetch, decode, etc. Each of these steps are, in turn, made up of a smaller series of steps called micro-operations.

- Micro-operation execution: Each step of the instruction cycle can be decomposed into micro-operation primitives that are performed in a precise time sequence. Each micro-operation is initiated and controlled based on the use of control signals / lines coming from the control unit.

  - Controller the data to move from one register to another

  - Controller the activate specific ALU functions

- Micro-instruction: Each instruction of the processor is translated into a sequence of lower-level micro-instructions. The process of translation and execution are to as microprogramming
- Microprogramming: The concept of microprogramming was developed by Maurice Wilkes in 1951, using diode matrices for the memory element. A micro program consist of a sequence of micro-instructions in a microprogramming.
- Micro programmed Control Unit is a relatively logic circuit that is capable of sequencing through micro-instructions and generating control signal to execute each micro-instruction.
- Control Unit: The control Unit is an important portion of the processor.

The control unit issues control signals external to the processor to cause data echange with memory and I/O unit. The control Unit issues also control signals internal to the processor to move data between registres, to perform the ALU and other internal operations in processor. In a hardwired control unit, the control signals are generated by a micro-instruction are used to controller register transfers and ALU operations. Control Unit design is then the collection and the implementation of all of the needed control signals for the micro-instruction executions.

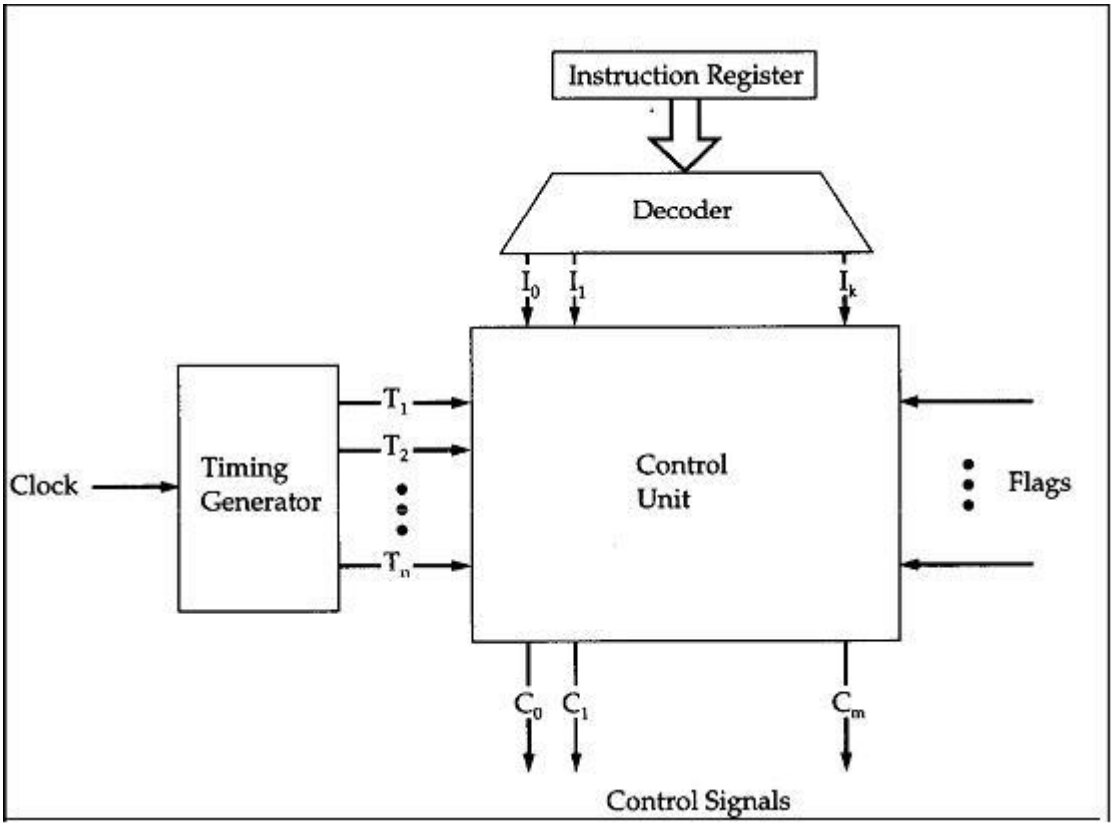
Control unit design approaches

How can we use the concept of microprogramming to implement a Control Unit ? There are two approaches of Control Unit Design and implementation:

- Micro programmed implementation

- Hardwired logic implementation

The figure 7.2 illustrated the control unit inputs. Two techniques have been used to implemented the CU. In a hardwired implementation, the control unit is essentially considered as a logic circuit. Its input logic signals are transformed into the set of output logic signals, which are the control signals. The approach of micro programmed implementation is studied in this section.



Approach of micro programmed control unit

Principe:

- The control signal values for each micro operation are stored in a memory.

- Reading the contents of the control store in a prescribed order is equivalent to sequencing through the micro operations

- Since the "micro program" of micro operations and their control signal values are stored in memory, this is a micro programmed unit.

Remarks:

- Are more systematic with a well defined format?
- Can be easily modified during the design process?
- Require more components to implement
- Tend to be slower than hardwired units (due to having to perform memory read operations)

Approach of hardwired logic

Principe:

- The Control Unit is viewed and designed as a combinatorial and sequential logic circuit.

- The Control Unit is implemented by using any of a variety of "standard" digital logic techniques. The logic circuit generate the fixed sequences of control signals

- This approach is used to generate fixed sequences of control signals with the higher speed.

Remarks:

- The principle advantages are a high(er) speed operation and the smaller implementations (component counts)
- The modifications to the design can be hard to do
- This approach is favored in RISC style designs

Micro programmed Control Unit

The ideal of micro programmed Control Unit is that the Control Unit design must include the logics for sequencing through micro-operations, for executing micro-operation, for executing micro-instructions, for interpreting opcodes and for making decision based on ALU flags. So the design is relatively inflexible. It is difficult to change the design if one wishes to add a new machine instruction.

The principal disadvantage of a micro programmed control unit is that it will be slower than hardwired unit of comparable technology. Despite this, microprogramming is the dominant technique for implementing control unit in the contemporary CISC processor, due to its ease of implementation.

The control unit operates by performing consecutive control storage reads to generate the next set of control function outputs. Performing the series of control memory accesses is, in effect, executing a program for each instruction in the machine's instruction set -- hence the term microprogramming.

The two basic tasks performed by a micro programmed control unit are as follows:

- Micro-instruction sequencing: the micro programmed control unit get the next mico-instruction from the control memory

- Micro-instruction execution: the micro programmed control unit generate the control signals needed to execute the micro-instruction.

The control unit design must consider both affect the format of the micro-instruction and the timing of the control unit.

Micro-instruction Sequencing

Two problems are involved in the design of a micro-instruction sequencing technique is the size of micro-instruction and the address-generation time. The first concern is obvious minimizing the size of the control memory. The second concern is simply a desire to execute microinstruction as fast as possible.

In executing a micro program, the address of the next microinstruction to be executed is one of these categories:

- Determined by instruction register

- Next sequential address

- Branch.

It is important to design compact time-efficient techniques for micro-instruction branching.

- Sequencing technique

Three general categories for a control memory address are as follows:

- Two address fields

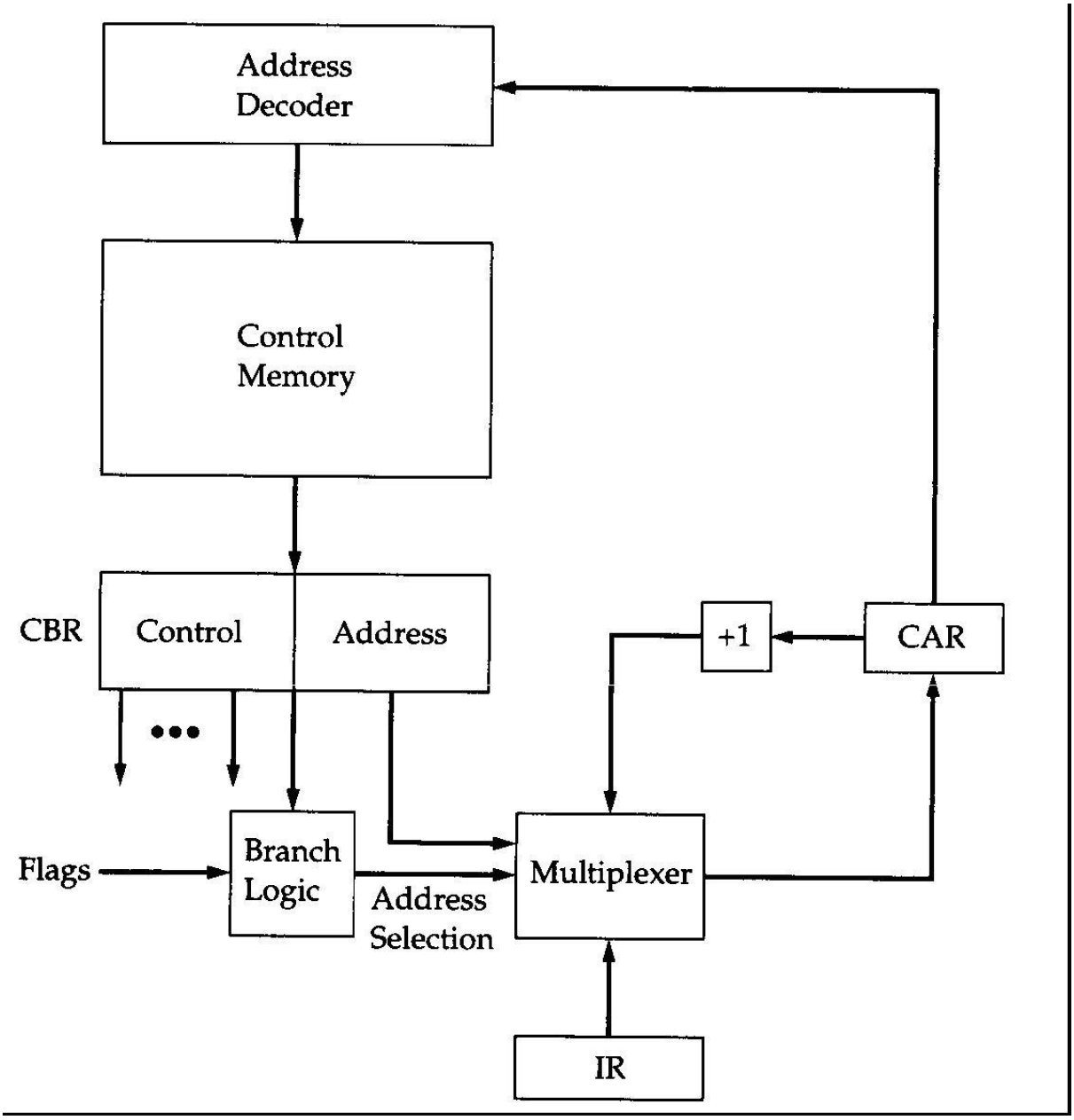
- Single address field

- Variable format

In Figure, the branch control logic with a single address field is illustrated.

Applications and advantages of microprogramming, Limitations of microprogramming:

What are the advantages and disadvantages of micro programmed control unit compared to hardwired control unit? Explain why hardwired control unit is usually used for RISC and micro programmed control unit for CISC architectures. Advantages:-The decoders and sequencing logic unit of a micro-programmed control unit are very simple pieces of logic, compared to the hardwired control unit, which contains complex logic for sequencing through the many micro-operations of the instruction cycle. It simplifies the design of the control unit. Simpler design means the control unit is cheaper and less error-prone to implement-It is also flexible as changes could be easily made to the design Principal Disadvantage:-Slower than a hardwired unit of comparable technology Hardwired control unit is used for RISC Architecture because hardwired is faster and can improve the performance Micro programmed control unit is used for CISC because it makes the design simpler and usually in CISC architecture, due to huge number of instructions in the instruction set, the control unit is quite complex. Hence it justifies using micro programmed control unit



Micro-instruction Execution

The microinstruction cycle is the basic event on a microprogrammed processor. Each cycle is made up the two parts: fetch and execute. This section deals with the execution of microinstruction. The effect of the execution of a microinstruction is to generate control signals for both the internal control to processor and the external control to processor.

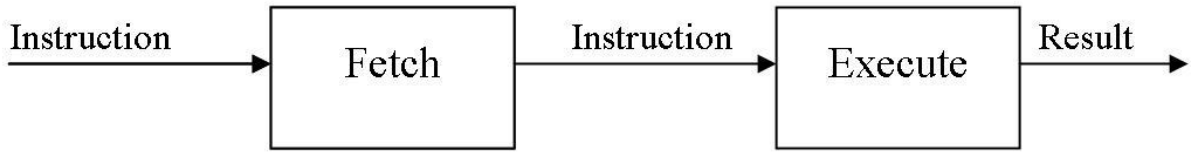
A organization of a control unit is shown in

Parallel Organization:

Pipelining

Basic concepts

An instruction has a number of stages. The various stages can be worked on simultanously through various blocks of production. This is a pipeline. This process is also referred as instruction pipeling. Figure 8.1 shown the pipeline of two independent stages: fetch instruction and execusion instruction. The first stage fetches an instruction and buffers it. While the second stage is executing the instruction, the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction. This process will speed up instruction execution



Pipeline principle

The decomposition of the instruction processing by 6 stages is the following.

- Fetch Instruction (FI): Read the next expected introduction into a buffer

- Decode Instruction (DI): Determine the opcode and the operand specifiers

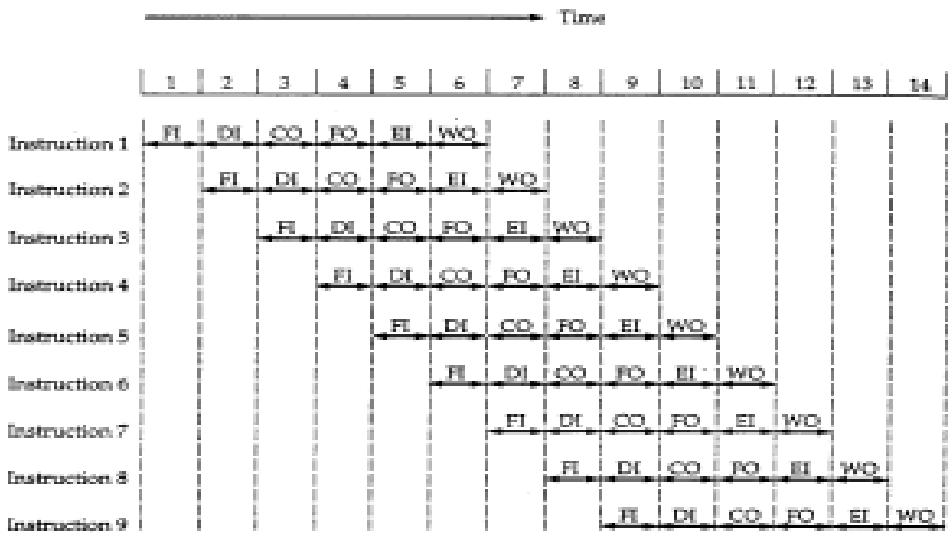
- Calculate Operands (CO): Calculate the effective address of each source operand. This may involve displacement, register indirect, indirect or other forms of address calculations.

- Fetch Operands (FO): Fetch each operand from memory. Operands in register need not be fetched.

- Execute Instruction (EI): Perform the indicated operation and store the result, if any, in the specified destination operand location.

- Write Operand (WO): Store result in memory.

Using the assumption of the equal duration for various stages, the figure 8.2 shown that a six stage pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units.



Also the diagram assumes that all of the stages can be performed in parallel, in particular, it is assumed that there are no memory conflicts. The processor make use of instruction pipelining to speed up executions, pipeling invokes breaking up the instruction cycle into a number of separate stages in a sequence. However the occurrence of branches and independencies between instruction complates the design and use of pipeline.

Pipeline Performance and Limitations

With the pipeling approach, as a form of parallelism, a “good” design goal of any system is to have all of its components performing useful work all of the time, we can obtain a high efficiency. The instruction cycle state diagram clearly shows the sequence of operations that take place in order to execute a single instruction.

This strategy can give the following:

- Perform all tasks concurrently, but on different sequential instructions

– The result is temporal parallelism.

– Result is the instruction pipeline.

Instruction Set Architecture (ISA):

Machine Instruction Characteristics

What is an Instruction Set?

From the designer's point of view, the machine instruction set provides the functional requirements for the CPU: Implementing the CPU is a task that in large part involves implementing the machine instruction set.

From the user's side, the user who chooses to program in machine language (actually, in assembly language) becomes aware of the register and memory structure, the types of data directly supported by the machine, and the functioning of the ALU.

Elements of an Instruction

Each instruction must have elements that contain the information required by the CPU for execution. These elements are as follows

- Operation code: Specifies the operation to be performed (e.g.. ADD, I/O). The operation is specified by a binary code, known as the operation code, or opcode.
- Source operand reference: The operation may involve one or more source operands, that is, operands that are inputs for the operation.
- Result operand reference: The operation may produce a result.
- Next instruction reference: This tells the CPU where to fetch the next instruction after the execution of this instruction is complete.

The next instruction to be fetched is located in main memory or, in the case of a virtual memory system, in either main memory or secondary memory (disk). In most cases, the next instruction to be fetched immediately follows the current instruction. In those cases, there is no explicit reference to the next instruction. Source and result operands can be in one of three areas:

- Main or virtual memory: As with next instruction references, the main or virtual memory address must be supplied.
- CPU register: With rare exceptions, a CPU contains one or more registers that may be referenced by machine instructions. If only one register exists, reference to it may be implicit. If more than one register exists, then each register is assigned a unique number, and the instruction must contain the number of the desired register.
- I/O device: The instruction must specify (he I/O module and device for the operation. If memory-mapped I/O is used, this is just another main or virtual memory address.

Instruction Cycle State Diagram

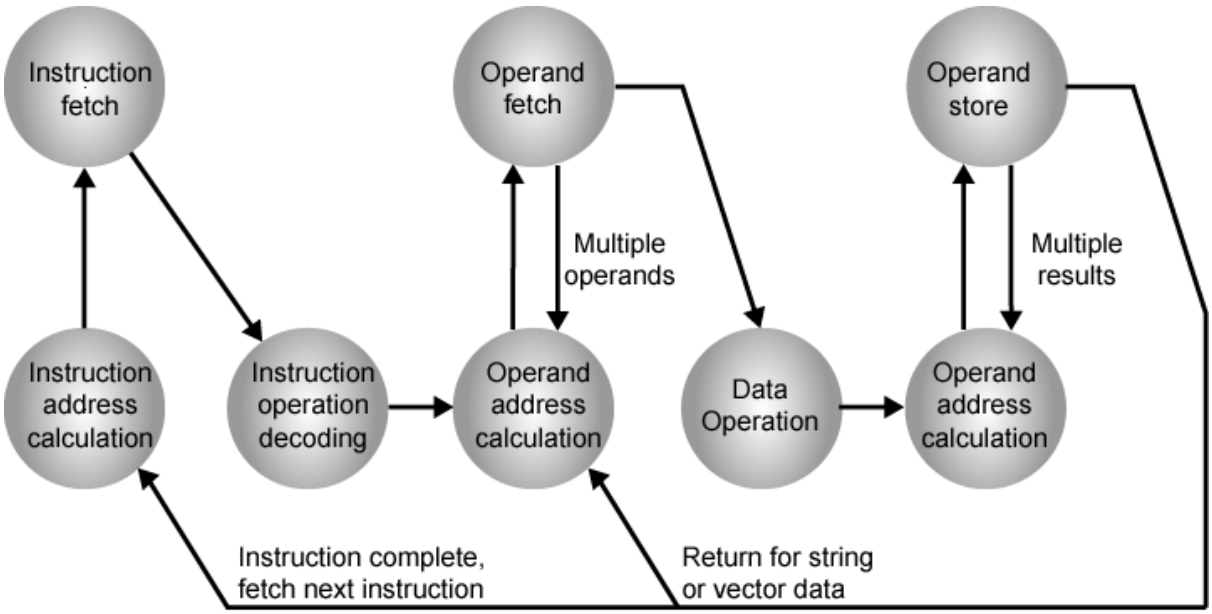


Figure 10.1 Instruction Cycle State Diagram

Instruction Representation

Within the computer, each instruction is represented by a sequence of bits. The instruction is divided into fields, corresponding to the constituent elements of the instruction. During instruction execution, an instruction is read into an instruction register (IR) in the CPU. The CPU must be able to extract the data from the various instruction fields to perform the required operation.

It is difficult for both the programmer and the reader of textbooks to deal with binary representations of machine instructions. Thus, it has become common practice to use a symbolic representation of machine instructions. Opcodes are represented by abbreviations, called mnemonics, that indicate the operation. Common examples include

ADD	Add
SUB	Subtract
MPY	Multiply
DIV	Divide
LOAD	Load data from memory
STOR	Store data to memory

Operands are also represented symbolically. For example, the instruction

ADD R, Y

may mean add the value contained in data location Y to the contents of register R. In this example. Y refers to the address of a location in memory, and R refers to a particular register. Note that the operation is performed on the contents of a location, not on its address.

Simple Instruction Format

RISC and CISC: Characteristics of CISC:

Complex Instruction Set Architecture (CISC) –
The main idea is that a single instruction will do all loading, evaluating, and storing operations just like a multiplication command will do stuff like loading data, evaluating, and storing it, hence it's complex.

Bot **CISC:** The CISC approach attempts to minimize the number of instructions per program but at the cost of increase in number of cycles per instruction.

h approaches try to increase the CPU performance

Characteristic of CISC –

1. Complex instruction, hence complex instruction decoding.
2. Instructions are larger than one-word size.
3. Instruction may take more than a single clock cycle to get executed.
4. Less number of general-purpose registers as operation get performed in memory itself.
5. Complex Addressing Modes.

More Data types.

Characteristics of RISC:

Reduced Instruction Set Architecture (RISC) –
The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating, and storing operations just like a load command will load data, store command will store the data

RISC: Reduce the cycles per instruction at the cost of the number of instructions per program.
Characteristic of RISC –

1. Simpler instruction, hence simple instruction decoding.
2. Instruction comes undersize of one word.
3. Instruction takes a single clock cycle to get executed.
4. More number of general-purpose registers.
5. Simple Addressing Modes.
6. Less Data types.

Pipeline can be achieved.

RISC versus CISC:

- Earlier when programming was done using assembly language, a need was felt to make instruction do more task because programming in assembly was tedious and error-prone due to which CISC architecture evolved but with the uprise of high-level language dependency on assembly reduced RISC architecture prevailed.

Example – Suppose we have to add two 8-bit number:

- **CISC approach:** There will be a single command or instruction for this like ADD which will perform the task.
- **RISC approach:** Here programmer will write the first load command to load data in registers then it will use a suitable operator and then it will store the result in the desired location.

So, add operation is divided into parts i.e. load, operate, store due to which RISC programs are longer and require more memory to get stored but require fewer transistors due to less complex command.

Difference –

Focus on software	Focus on hardware
Uses only Hardwired control unit	Uses both hardwired and micro programmed control unit
Transistors are used for more registers	Transistors are used for storing complex Instructions
Fixed sized instructions	Variable sized instructions
Can perform only Register to Register Arithmetic operations	Can perform REG to REG or REG to MEM or MEM to MEM
Requires more number of registers	Requires less number of registers
Code size is large	Code size is small

An instruction execute in a single clock cycle	Instruction takes more than one clock cycle
An instruction fit in one word	Instructions are larger than the size of one word

Vector Processing:

Requirements and Characteristics of vector processing:

Vector processing performs the arithmetic operation on the large array of integers or floating-point number. Vector processing operates on all the elements of the array in parallel providing each pass is independent of the other.

Vector processing avoids the **overhead** of the loop control mechanism that occurs in general-purpose computers.

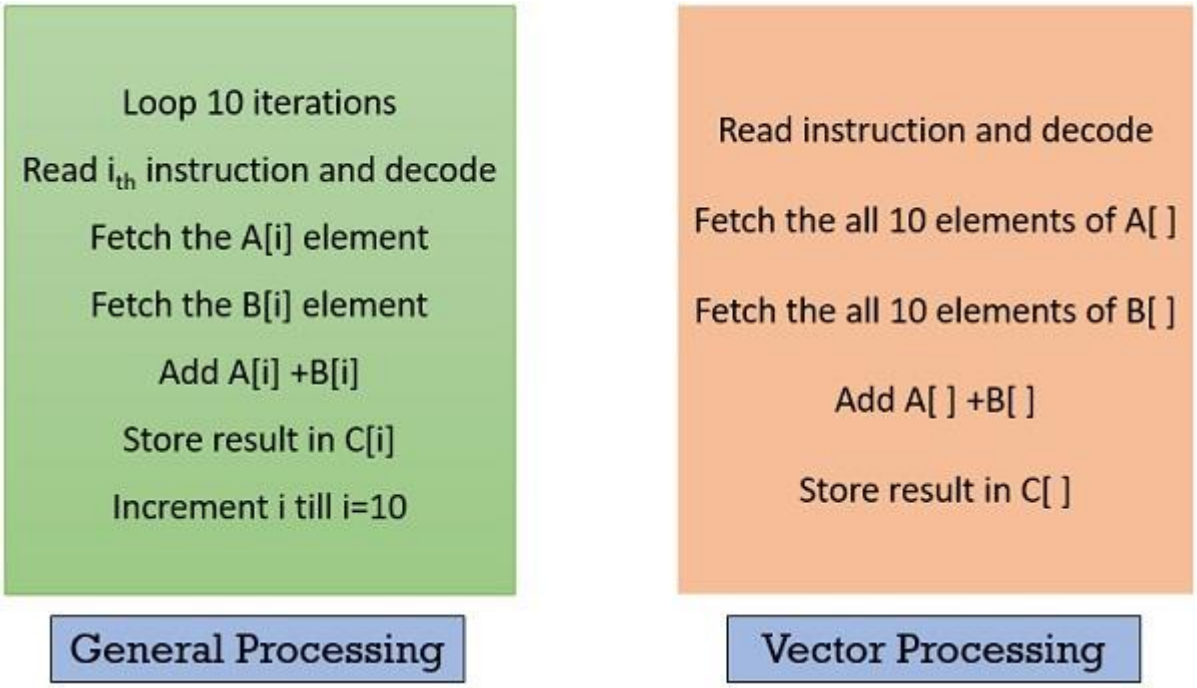
In this section, we will have a brief introduction on vector processing, its characteristics, about vector instructions and how the performance of the vector processing can be enhanced? So lets us start.

Introduction

We need computers that can solve mathematical problems for us which include, arithmetic operations on the large arrays of integers or floating-point numbers quickly. The general-purpose computer would use loops to operate on an array of integers or floating-point numbers. But, for large array using loop would cause overhead to the processor.

To avoid the overhead of processing loops and fasten the computation, some kind of parallelism must be introduced. **Vector processing** operates on the entire array in just one operation i.e. it operates on elements of the array in **parallel**. But, vector processing is possible only if the operations performed in parallel are **independent**.

Look at the figure below, and compare the vector processing with the general computer processing, you will notice the difference. Below, instructions in both the blocks are set to add two arrays and store the result in the third array. Vector processing adds both the array in parallel by avoiding the use of the loop.



Operating on multiple data in just one instruction is also called **Single Instruction Multiple Data** (SIMD) or they are also termed as **Vector instructions**. Now, the data for vector instruction are stored in **vector registers**.

Each vector register is capable of storing several data elements at a time. These several data elements in a vector register is termed as a **vector operand**. So, if there are n number of elements in a vector operand then n is the **length of the vector**.

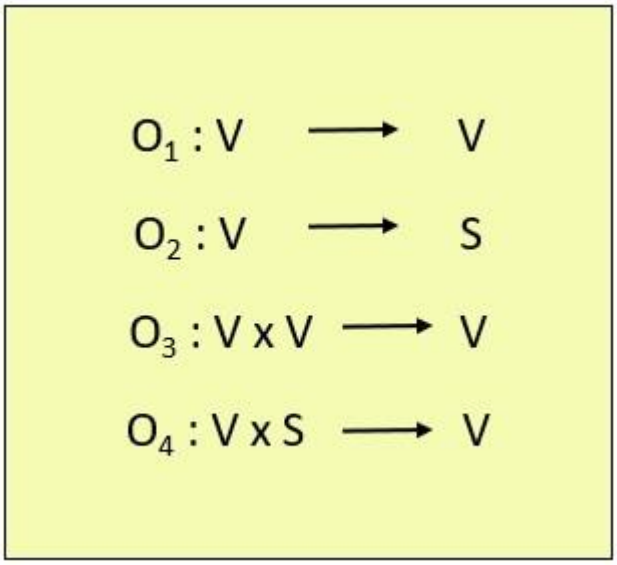
Supercomputers were evolved to deal with billions of floating-point operations/second. Supercomputer optimizes numerical computations (vector computations).

But, along with vector processing supercomputers are also capable of doing scalar processing. Later, **Array processor** was introduced which particularly deals with vector processing, they do not indulge in scalar processing.

Characteristics of Vector Processing

Each element of the vector operand is a **scalar quantity** which can either be an integer, floating-point number, logical value or a character. Below we have classified the vector instructions in four types.

Here, V is representing the vector operands and S represents the scalar operands. In the figure below, O1 and O2 are the unary operations and O3 and O4 are the binary operations.



Most of the vector instructions are **pipelined** as vector instruction performs the same operation on the different data sets repeatedly. Now, the pipelining has start-up delay, so longer vectors would perform better here.

The pipelined vector processors can be classified into two types based on from where the operand is being **fetched** for vector processing. The two architectural classifications are Memory-to-Memory and Register-to-Register.

In **Memory-to-Memory** vector processor the operands for instruction, the intermediate result and the final result all these are retrieved from the **main memory**. TI-ASC, CDC STAR-100, and Cyber-205 use memory-to-memory format for vector instructions.

In **Register-to-Register** vector processor the source operands for instruction, the intermediate result, and the final result all are retrieved from **vector or scalar registers**. Cray-1 and Fujitsu VP-200 use register-to-register format for vector instructions.

Vector Instruction

A vector instruction has the following fields:

1. Operation Code

Operation code indicates the **operation that** has to be performed in the given instruction. It decides the functional unit for the specified operation or reconfigures the multifunction unit.

2. Base Address

Base address field refers to the **memory location** from where the operands are to be fetched or to where the result has to be stored. The base address is found in the memory reference instructions. In the vector instruction, the operand and the result both are stored in the vector registers. Here, the **base address** refers to the designated **vector register**.

3. Address Increment

A vector operand has several data elements and address increment specifies the **address of the next element in the operand**. Some computer stores the data element consecutively in main memory for which the increment is always 1. But, some computers that do not store the data elements consecutively requires the variable address increment.

4. Address Offset

Address Offset is always specified related to the base address. The effective **memory address** is calculated using the address offset.

5. Vector Length

Vector length specifies the **number of elements in a vector operand**. It identifies the **termination** of a vector instruction.