CHAPTER ONE

Introduction to the Microprocessors and Computers

1.1 The microprocessor-based personal computer system

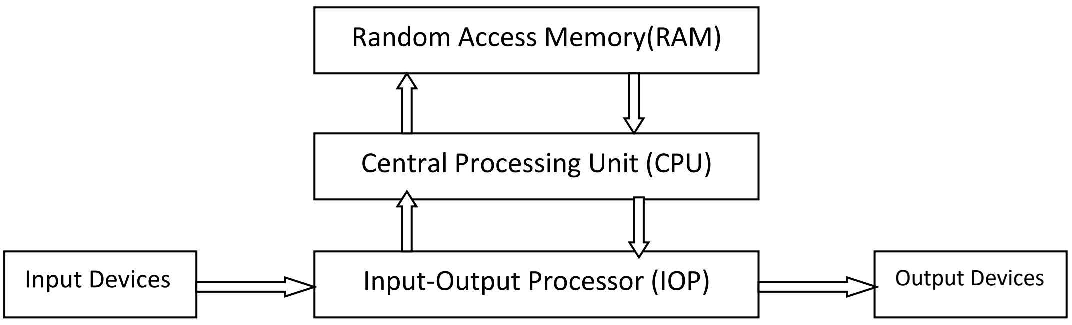
The computer lies at the heart of computing. Without it most of the computing disciplines today would be a branch of theoretical mathematics. To be a professional in any field of computing today, one should acquire some understanding and appreciation of a computer system's functional components, their characteristics, their performance, and their interactions. We need to understand computer architecture in order to structure a program so that it runs more efficiently on a real machine. In selecting a system to use, they should be able to understand the trade-off among various components, such as CPU clock speed vs. memory size.

Digital computers use the binary number system, which has two digits; 0 and 1. Because of the physical restriction of components i.e., inside the computer, there are integrated circuits with thousands of transistors. These transistors are made to operate on two-state**.** By this design, all the input and output voltages are either HIGH or LOW. Low voltage represents binary 0 and high voltage represents binary 1.

A binary digit is called a **bit**. Information is represented in digital computers in groups of bits. By using various coding techniques, groups of bits can be made to represent not only binary numbers but also other discrete symbols, such as decimal digits or letters of the alphabet and to develop complete sets of instructions for performing various types of computations.

A computer system is subdivided into two functional entities: Hardware and Software. The hardware of the computer consists of all the electronic components and electromechanical devices that comprise the physical entity of the device. Computer software consists of the instructions and data that the computer manipulates to perform various data-processing tasks. A sequence of instructions for the computer is called a **Program**.

The hardware of the computer is usually divided into three major parts.


Random Access Memory(RAM)
Central Processing Unit (CPU)
Input Devices
Input-Output Processor (IOP)
Output Devices

--Block diagram of a digital computer—

- **The central processing unit (CPU)** contains an **arithmetic and logic** unit for manipulating data, a number **of registers** for storing data, and **control circuits** for fetching and executing instructions.
- **The memory** of a computer contains storage for instructions and data. It is called a **Random Access Memory(RAM)** because the CPU can access any location in memory at random and retrieve the binary information within a fixed interval of time.
- **The Input and output processor (IOP)** contains electronic circuits for communicating and controlling the transfer of information between the computer and the outside world.
- **The input and output devices** connected to the computer include keyboards, printers, terminals, and other communication devices.

Computer Architecture

Those attributes of the system that is visible to a programmer**. It is concerned with the structure and behavior of the computer as seen by the user.** Those attributes that have a direct impact on the execution of a program.

• Instruction sets

• Data representation – number of bits used to represent data

• Input/output mechanisms

• Memory addressing techniques

Computer Organization :

The operational units and their interconnections that realize the architectural specifications. **It is concerned with the way the hardware components operate and the way they are connected together to form the computer system.** Those hardware attributes that are transparent to the programmer.

• Control signals

• Interfaces between the computer and peripherals

• Memory technology

Computer Design

It is concerned with hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system. This is sometimes referred to as computer implementation.

1.2 Review of number systems

DATA TYPES

Binary information is stored in **memory** or **processor registers.**

Registers contain either **data** or **control information.**

- **Data** are numbers and other binary-coded information
- **Control information** is a bit or a group of bits used to specify the sequence of command signals

Data types found in the registers of digital computers

- **Numbers** used in arithmetic computations
- **Letters** of the alphabet used in data processing

  Other discrete symbols used for specific purpose

Number Systems

- ✓ Most modern computer systems do not represent numeric values using the decimal system.
- ✓ Numbering system it uses a specific radix (base).
- ✓ Radices that are power of 2 are widely used in digital systems.
- ✓ These radices include

Binary (base 2),

Quaternary (base 4),

Octagonal (base 8),

Decimal (base 10) and

Hexagonal (base 16).

- ✓ The base 2 binary system is dominant in computer systems.

Most common number system: Decimal, Binary, Octal, Hexadecimal

Decimal number system is important because it is universally used to represent quantities outside a digital system. Decimal system uses numbers from 0 to 9. This is to the base 10.

- ➢ In everyday life we use a system based on decimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) to represent numbers and refer to the system as the decimal system.

Example: the number 83 means. It means eight tens plus three: 83 = (8 * 10) + 3

- ➢ The decimal system is said to have a **base, or radix, of 10. This means that each** digit in the number is multiplied by 10 raised to a power corresponding to that digit's position:
- ➢ The same principle holds for decimal fractions but negative powers of 10 are used.

Example: the decimal fraction 0.256 stands for 2 tenths plus 5 hundredths plus 6 thousandths ($2*10^{-1} + 5*10^{-2} + 6*10^{-3}$)

Binary numbers are based on the concept of ON or OFF. Binary number system has a base of 2. Its two digits are denoted by 0 & 1 and are called bits.

- In the decimal system, 10 different digits are used to represent numbers with a base of 10.
- In the binary system, we have only two digits, 1 and 0. Thus, numbers in the binary system are represented to the base 2.

Octal Number system uses exactly eight symbols 0,1,2,3,4,5,6, and 7. i.e., it has a base of 8. Each octal digit has a unique 3 bit binary representation.

Hexadecimal number system is to the base 16. It spans from 0 to 9 and then A to E. In hexadecimal system A,B,C,D,E and F represents 10,11,12,13,14 and 15, i.e., it has a base of 16. Hexadecimal numbers are more convenient for people to recognize and interpret than the long strings of binary numbers

Decimal	Binary Code	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000		8
9	1001		9
10	1010		A
11	1011		B
12	1100		C
13	1101		D
14	1110		E
15	1111		F

Conversion Of Number System

The right most bit is called Least Significant Bit (LSB) and left Most Bit is called Most Significant Bit (MSB).

Ie.,

$$\begin{tikzcd} & 100111 \arrow[dl] \arrow[dr] & \\ \text{MSB} & & \text{LSB} \end{tikzcd}$$

Conversion from Binary to Decimal

The binary number system has a base 2, the position weights are used on the power of 2.

To convert a binary number to a decimal numbers system, we follow expanding method.

Ex: 100111 = ?

$1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$1 \times 32 + 0 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 39$

100111 = 39 → Ans.

Ex- 2: 0.1010 = ?

$1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4}$

$1 \times (1/2) + 0 \times (1/4) + 1 \times (1/8) + 0 \times (1/16)$

$1 \times 0.5 + 0 \times 0.25 + 1 \times 0.125 + 0 \times 0.0625$

$(8 + 4 + 0 + 1) + (0.5 + 0 + 0.125 + 0) = 13.625$

0.1010 = 0.625 → Ans.

Ex-3: 1101.1010 = ?

$(1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0) + (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4})$

$(1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1) + (1 \times 0.5 + 0 \times 0.25 + 1 \times 0.125 + 0 \times 0.0625)$

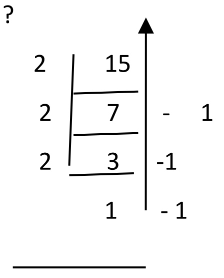
$(8 + 4 + 0 + 1) + (0.5 + 0 + 0.125 + 0) = 13.625$

1101.1010 = 13.625 → Ans.

<u>Conversion from Decimal to Binary</u>

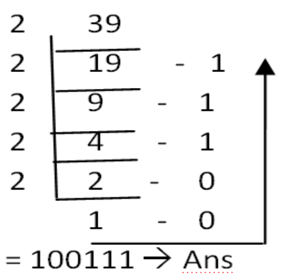
A positive integer can be easily converted to equivalent binary form by the repeated division by **2**. Start by dividing the given decimal integer by **2**. Let **r1** be the remainder and **q1** the quotient. Divided **q1** by **2** and let **r2** be the remainder and **q2** be the quotient. Continue the process of division by **2** until **0** is obtained as a quotient. Reading the remainders from the bottom to the first top can form the equivalent binary number.

Ex: 15 = ?



15 = 1111 → Ans

Ex 2 : 39 = ?



39 = 100111 → Ans

Conversion from Decimal to Octal:

```
Ex:   3977 = ?
        8|3977
        8|497   - 1
        8|62   - 1
           7   - 6
```

3977 = 7611 → Ans

Conversion from Octal to Decimal:

The binary number system has a base 8, the position weights are used on the power of 8.

Ex: 3077 = ?

$3 \times 8^3 + 0 \times 8^2 + 7 \times 8^1 + 7 \times 8^0$

$3 \times 512 + 0 \times 64 + 7 \times 8 + 7 \times 1 = 1599$

3077 = 1599 → Ans.

Conversion from Binary to Octal :

Ex: 10011.11001 = ?

→ 010 011 . 110 010

→ 2 3 6 2

= 23.62

Conversion from Octal to Binary :

Ex1: 37

= 011 111

=011111

Ex2: 377.77

011 111 111 . 111 111

Conversion from Decimal to Hexadecimal:

Ex: 39 = ?

```
16 |39
    2  - 7
```

39 = 27 → Ans

Ex: 256 = ?

```
16| 256
16| 16   - 0
     1   - 0
```

256 = 100 → Ans

Conversion from Hexadecimal to Decimal:

The binary number system has a base 16, the position weights are used on the power of 16.

Ex: 39A = ?

$3 \times 16^2 + 9 \times 16^1 + A \times 16^0$

$3 \times 256 + 9 \times 16 + A \times 1$

$3 \times 256 + 9 \times 16 + 10 \times 1 = 922$ (i.e., decimal 10 for A Hex)

39A = 922 → Ans.

Conversion from Binary to Hexadecimal :

Ex: 11100101110111

```
=0011  1001  0111    0111
=  3     9     7       7
```

=3977

Ex: 110101001.0010110

```
=0001   1010   1001   .  0010   1100
=1        A      9    .    2      C
```

Conversion from Hexadecimal to Binary :

Ex: 39A

=0011 1001 1010

BINARY ARITHMETIC

Computer understands only the language of binary numbers. Therefore, the machine performs what is called binary arithmetic (binary computation).

Binary addition

Binary addition operates by the same rule as decimal addition, except that it is simpler. A carry to the next higher order (or more significant) position occurs when the sum is decimal 2, that is, binary 10. Therefore, the binary addition rules may be written as follows:

0+0=0

0+1=1

1+0=1

1+1=0 plus a carry of 1 into the next position

1+1+1=1 plus a carry of 1 into the next position.

The last case occurs when the two binary digits in a certain position are 1s and there is a carry from the previous position.

Example1:

6+7 =13

110+111=1101

Binary Subtraction

It operates by the same rule as decimal subtraction. The rule is as follows;

0-0=0 1-0=1 1-1=0 10-1=1

Example:

$$\begin{array}{rrr} 11100 & 28 & \\ -\ \underline{11010} & \underline{-26} & \\ \underline{\underline{00010}} & & =2 \end{array} \qquad \begin{array}{rrr} 101101 & 45 & \\ -\ \underline{111} & \underline{-7\quad} & \\ \underline{\underline{100110}} & & =38 \end{array}$$

Representation Of Negative Numbers

There are different ways of representing negative numbers in a computer.

Sign- magnitude representation

In signed binary representation, the left-most bit is used to indicate the sign of the number. Traditionally, **0** is used to denote a positive number and **1** is used to denote a negative number.

In a 5- bit representation we use the first bit for sign and the remaining 4- bits for the magnitude.

In the binary number system there are two types of complements, 1's complement and 2's complement. To represent a negative binary number 2's complement is most widely used.

1's complement

To get 1's complement of a binary number each bit of the binary number is subtracted from 1.

The 1's complement of 01 is 10. Thus we can see that the 1's complement of a binary number can be obtained by simply changing each bit 1 to 0 and 0 to 1.

Ex1 : Find 1's complement of 1100101

The 1's complement is 0011010

Ex2: +2 is 00000010

-2 is 11111101

Note that in this representation positive numbers start with a **0** on the left, and negative numbers start with a **1** on the left most bit.

2's Complement:

The 2's complement of a binary number is equal to the 1's complement of the number plus one.

Ex: 11010 $_{(2)}$ =? 2's complement

1's complement of 11010 is 00101

$$\begin{array}{r} 00101 \\ +\quad 1 \\ \hline \mathbf{00110} \\ \hline \end{array}$$

⇨ 11010 $_{(2)}$ = **00110 (2's complement)**

Ex2: 0000 (2) = ? 2's complement

1's complement of 000 is 1111

$$\begin{array}{r} 1111 \\ 1 \\ \hline \underline{\mathbf{10000}} \end{array}$$

0000 => **1** **0000** (here 1 is carry)

Arithmetic Addition:

The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. If the signs are same, we add the two magnitudes and give the sum the common sign. If the sign are different, we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude.

Add the two numbers, including their sign bits, and discard any carry out of the sign bit position. Numerical examples for addition are shown below. Note that negative numbers must initially be in 2's complement and that if the sum obtained after the addition is negative, it is in 2's complement form.

```
+6     00000110    -6     11111010
+13    00001101    +13    00001101
-----------------         --------
+19    00010011    +7     00000111
```

In each of the two cases, the operation performed is always addition, including the sign bits. Any carry out of the sign bit position is ***discarded***, and negative results are automatically in 2's complement form.

Arithmetic subtraction:

Subtraction of two signed binary numbers when negative numbers are in 2's complement form is very simple.

The complement of a negative number in complement form produces the equivalent positive number.

Ex: (-6) – (-13) = +7

In binary with 8 bits this is written as 11111010 – 11110011.

The subtraction is changed to addition by taking the 2's complement of (-13) to give (+13).

Ie., 11110011 → 2's complement is → 0000 1101

In binary this is 11111010 + 00001101 = 100000111.

Removing the end carry, we obtain the correct answer 00000111 (+7)

It is worth noting that binary numbers in the signed 2's complement system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers.

Therefore, computers need only one common hardware circuit to handle both types of arithmetic.

Overflow:

When two numbers of n digits each are added and the sum occupies n+1 digits, we say that an overflow occurred.

An overflow is a problem in digital computers because the width of registers is finite.

A result that contains n+1 bits cannot be accommodated in a register with a standard length of n bits. For this reason, many computers detect the occurrence of an overflow, and when it occurs, a corresponding flip-flop is set which can then be checked by the user.

An overflow may occur if the two numbers added are both positive or both negative.

Example: Two signed binary numbers, +70 and +80, are stored in two 8bit registers.

carries :	**0** **1**		**carries:**	**1** **0**	
+70	0	1000110	-70	1	0111010
+80	0	1010000	-80	1	0110000
+150	1	0010110	-150	0	1101010

Note that the 8 bit result that should have been positive has a negative sign bit and the 8bit result that should have been negative has a positive sign bit. If, however, the carry out of the sign bit position is taken as the sign bit of the result, the 9 bit answer so obtained will be correct.

Since the answer cannot be accommodated within 8 bits, we say that an overflow occurred. If the two carries are applied to an exclusive-OR gate, an overflow will be detected when the output of the gate is equal to 1.

Chapter Two
The microprocessor and its architecture

2. Inside the Intel 8086

2.1.1. Von Neumann architecture

The so-called **von Neumann architecture** is a model for a computing machine that uses a single storage structure to hold both the set of instructions on how to perform the computation and the data required or generated by the computation. Such machines are also known as **stored-program computers**. The separation of storage from the processing unit is implicit in this model.

The architecture is named after mathematician John von Neumann who provided an early written account of a general purpose stored-program computing machine.

By treating the instructions in the same way as the data, a stored-program machine can easily change the instructions. In other words the machine is reprogrammable. One important motivation for such a facility was the need for a program to increment or otherwise modify the address portion of instructions. This became less important when index registers and indirect addressing became customary features of machine architecture.

The Von Neumann bottleneck: CPU and Memory separation is implicit. Trends in high improvement on CPU performance but lagging performance in memory degrade overall performance.

2.1.2. Harvard Architecture

The term **Harvard architecture** originally referred to computer architectures that used physically separate storage and signal pathways for their instructions and data (in contrast to the *von Neumann architecture*). The term originated from the Harvard Mark I relay-based computer, which stored instructions on punched tape and data in relay latches.

In a computer with a von Neumann architecture, the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instructions and data use the same signal pathways and memory. In a computer with Harvard architecture, the CPU can read both an instruction and data from memory at the same time. A computer with Harvard architecture can be faster because it is able to fetch the next instruction at the same time it completes the current instruction. Speed is gained at the expense of more complex electrical circuitry.

In recent years the speed of the CPU has grown many times in comparison to the access speed of the main memory. Care needs to be taken to reduce the number of times main memory is accessed in order to maintain performance. If, for instance, every instruction run in the CPU requires an access to memory, the computer gains nothing for increased CPU speed - a problem referred to as being *memory bound*.

Memory can be made much faster, but only at high cost. The solution then is to provide a small amount of very fast memory known as a cache. As long as the data the CPU needs is in the cache, the performance hit is very much less than it is if the cache then has to turn around and get the data from the main memory. Tuning the cache is an important aspect of computer design.

Modern high performance CPU chip designs incorporate aspects of both Harvard and von Neumann architecture. On chip cache memory is divided into an instruction cache and a data cache. Harvard architecture is used as the CPU accesses the cache. In the case of a cache miss, however, the data is retrieved from the main memory, which is not divided into separate instruction and data sections. Thus a von Neumann architecture is used for off chip memory access.

MICROOPERATIONS

- The operations on the data in registers are called microoperations.
- The functions built into registers are examples of microoperations

  –Shift, Load, Clear, Increment, etc…

ORGANIZATION OF A DIGITAL SYSTEM

Definition of the (internal) organization of a computer:

- Set of registers and their functions
- Micro operations set
- Set of allowable microoperations provided by the organization of the computer
- Control signals that initiate the sequence of microoperations (to perform the functions)

REGISTER TRANSFER LEVEL

- Viewing a computer, or any digital system, in this way is called the register transfer level
- This is because we're focusing on
  - The system's registers
  - The data transformations in them, and
  - The data transfers between them.

REGISTER TRANSFER LANGUAGE

- Rather than specifying a digital system in words, a specific notation is used, *register transfer language*

- For any function of the computer, the register transfer language can be used to describe the (sequence of) microoperations
- Register transfer language
    - ✓ is a symbolic language
    - ✓ is a convenient tool for describing the internal organization of digital computers
    - ✓ Can also be used to facilitate the design process of digital systems

DESIGNATION OF REGISTERS

- Registers are designated by capital letters, sometimes followed by numbers (e.g., A, R13, IR)
- Often the names indicate function:
    - **MAR** -Memory Address register
    - **PC** -Program Counter
    - **IR** -Instruction Register
- Registers and their contents can be viewed and represented in *various ways*
- A register can be viewed as a single entity:

- Registers may also be represented showing the bits of data they contain

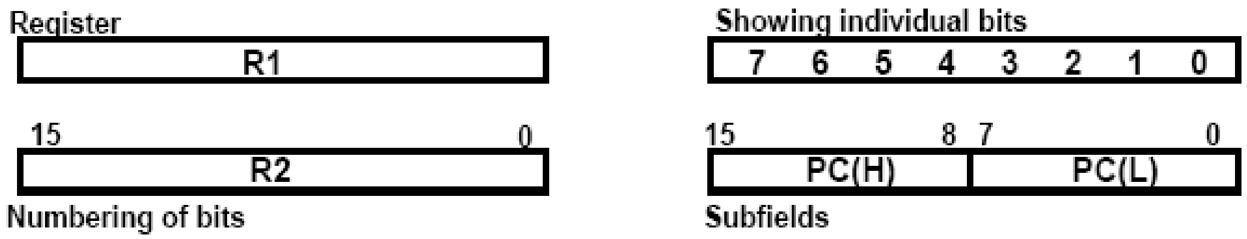
✓Designation of a register

- a register

-portion of a register

-a bit of a register

✓Common ways of drawing the block diagram of a register



REGISTER TRANSFER

✓Copying the contents of one register to another is a register transfer

• A register transfer is indicated as: **R2 ←R1**

- ✓ In this case the contents of register R1 are copied (loaded) into register R2
- ✓ A simultaneous transfer of all bits from the source R1 to the destination register R2, during one clock pulse

- ○ Notice that this is a non-destructive (i.e. the contents of R1 are not altered by copying (loading) them to R2)
- ○ A register transfer such as

R3 ←R5

- ○ Implies that the digital system has
  - ✓ the data lines from the source register (R5) to the destination register (R3)
  - ✓ Parallel load in the destination register (R3)
  - ✓ Control lines to perform the action

<u>CONTROL FUNCTIONS</u>

- ○ Often actions need to only occur if a certain condition is true

- This is similar to an "if" statement in a programming language
- In digital systems, this is often done via **a *control signal, called a control function***
- If the signal is 1, the action takes place
- This is represented as: **P: R2 ←R1**

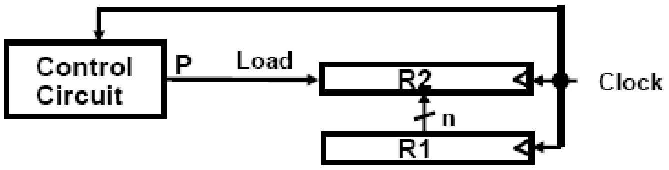
➢ Which means "if P = 1, then load the contents of register R1 into register R2" (i.e., if (P = 1) then (R2 ←R1))

<u>HARDWARE IMPLEMENTATION OF CONTROLLED TRANSFERS</u>

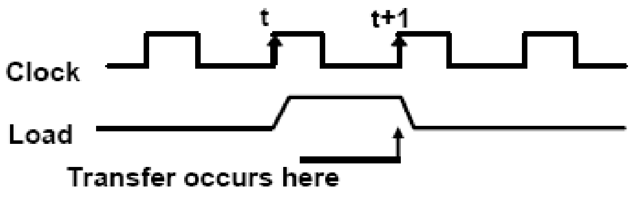
➢ Implementation of controlled transfer

➢ P: R2 ←R1

Block diagram



Timing diagram



- The same clock controls the circuits that generate the control function and the destination register.
- Registers are assumed to use *positive-edge-triggered flip-flops*

SIMULTANEOUS OPERATIONS

- If two or more operations are to occur simultaneously, they are separated with commas

  P: R3 $\leftarrow$R5, MAR $\leftarrow$IR
- Here, if the control function P = 1, load the contents of R5 into R3, and at the same time (clock), load the contents of register IR into register MAR

BASIC SYMBOLS FOR REGISTER TRANSFERS

Symbols	Description	Examples
Capital letters & numerals	Denotes a register	MAR, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow $\leftarrow$	Denotes transfer of information	$R2 \leftarrow R1$
Colon :	Denotes termination of control function	P:
Comma ,	Separates two micro-operations	$A \leftarrow B,\ B \leftarrow A$

Bus and Memory Transfer

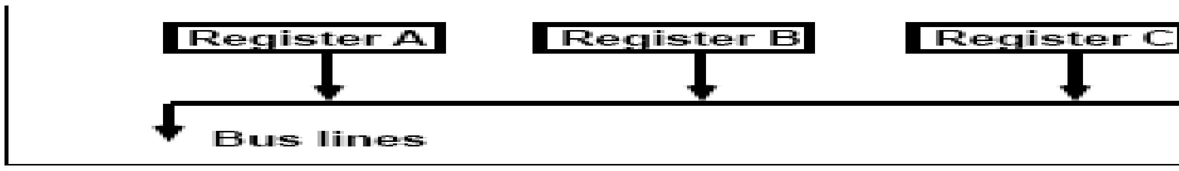
CONNECTING REGISTRS

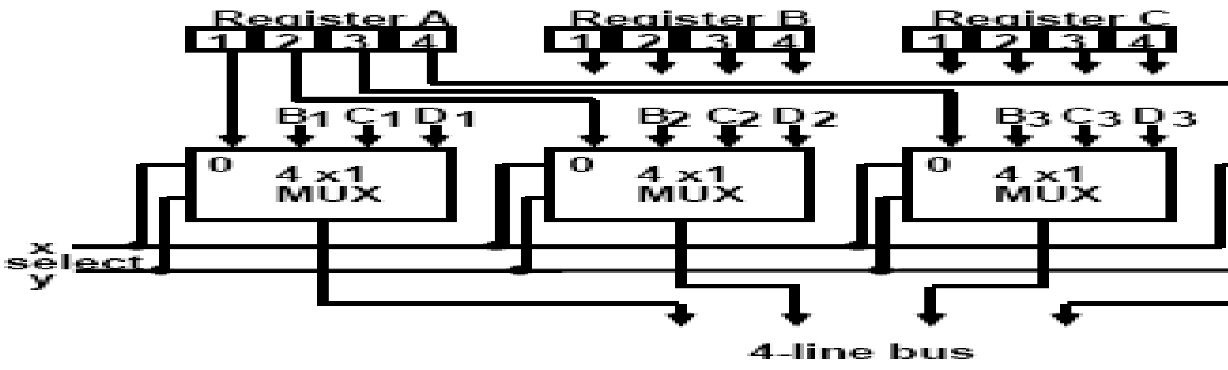
- In a digital system with many registers, it is impractical to have data and control lines to directly allow each register to be loaded with the contents of every possible other registers
- To completely connect n registers →n(n-1) lines, so the cost is in order $O(n^2)$ cost
- This is not a realistic approach to use in a large digital system
- **Instead, take a different approach**

- Have one centralized set of circuits for data transfer **–the bus**
- Have control circuits to select which register is the source, and which is the destination

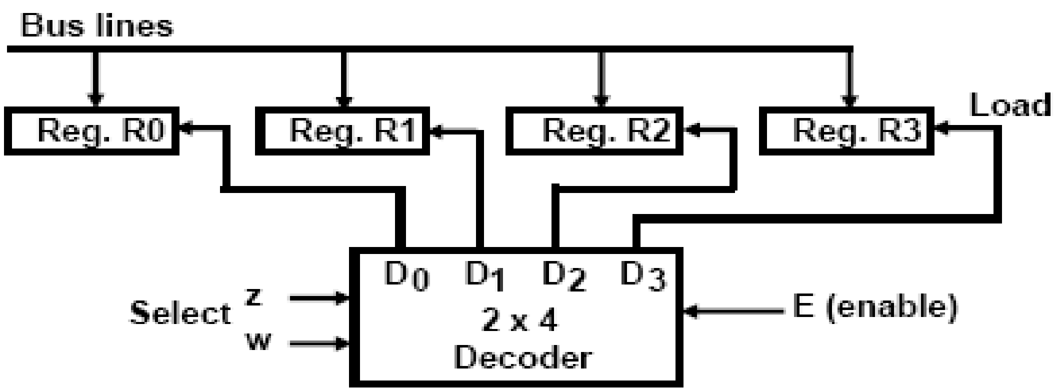
BUS:

- is a path(of a group of wires) over which information is transferred, from any of several sources to any of several destinations.
- From a register to bus: BUS ←R





TRANSFER FROM BUS TO A DESTINATION REGISTER



<u>BUS TRANSFER IN RTL</u>

- Depending on whether the bus is to be mentioned explicitly or not, register transfer can be indicated as either
  - **R2 ←R1**
  - **BUS ←R1, R2 ←BUS**
- In the former case the bus is implicit, but in the latter, it is explicitly indicated

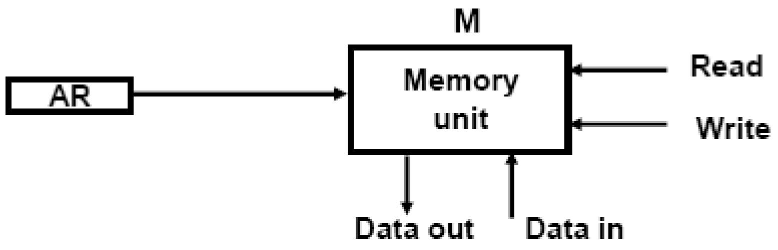
<u>MEMORY TRANSFER</u>

- ✓ Collectively, the memory is viewed at the register level as a device, M.
- ✓ Since it contains multiple locations, we must specify which address in memory we will be using
- ✓ This is done by indexing memory references

✓

- ✓ Memory is usually accessed in computer systems by putting the desired address in a special register, the *Memory Address Register(MAR, or AR)*

- ➢ When memory is accessed, the contents of the MAR get sent to the memory unit's address lines



MEMORY READ

- ➢ To read a value from a location in memory and load it into a register, the register transfer language notation looks like this:

  R1 ←M[MAR]

- ➢ This causes the following to occur

- ✓ The contents of the MAR get sent to the memory address lines
- ✓ A Read (= 1) gets sent to the memory unit
- ✓ The contents of the specified address are put on the memory's output data lines
- ✓ These get sent over the bus to be loaded into register R1

<u>MEMORY WRITE</u>

- To write a value from a register to a location in memory looks like this in register transfer language:

  M[MAR] ← R1

- This causes the following to occur
  - The contents of the MAR get sent to the memory address lines
  - A Write (= 1) gets sent to the memory unit

- The values in register R1 get sent over the bus to the data input lines of the memory
- The values get loaded into the specified address in the memory

MICROOPERATIONS

Computer system microoperations are of four types:

1. Register transfer microoperations
2. Arithmetic microoperations
3. Logic microoperations
4. Shift microoperations

ARITHMETIC MICROOPERATIONS

- The basic arithmetic microoperations are

  –Addition –Subtraction

  –Increment –Decrement

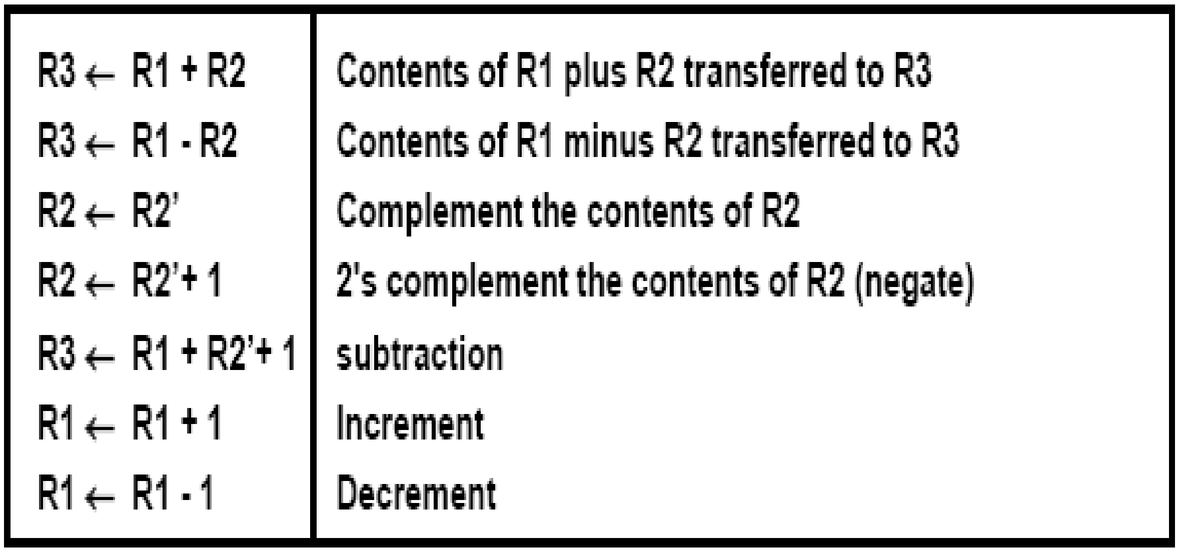
- The additional arithmetic microoperations are

  –Add with carry

  –Subtract with borrow

  –Transfer/Load etc. …

Summary of Typical Arithmetic Micro-Operations



$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow R2'$	Complement the contents of R2
$R2 \leftarrow R2' + 1$	2's complement the contents of R2 (negate)
$R3 \leftarrow R1 + R2' + 1$	subtraction
$R1 \leftarrow R1 + 1$	Increment
$R1 \leftarrow R1 - 1$	Decrement

CHAPTER THREE

INPUT/OUTPUT ORGANIZATION

I/O SUBSYSTEM (OR I/O)

The input output subsystem of a computer, referred to as I/O, provides an efficient mode of communication between the central system and the outside environment. Programs and data must be entered into computer memory for processing and results obtained from computations must be recorded or displayed for the user.

The CPU is an extremely fast device capable of performing operations at very high speed. When input information is transferred to the processor via a slow keyboard, the processor will be idle most of the time while waiting for the information to arrive. To use a computer efficiently, a large amount of programs and data must be prepared in advance and transmitted into a storage medium such as magnetic tapes or disks.

Peripheral Device (Or I/O Device)

Input or output devices attached to the computer are called **Peripherals**. Among the most common peripherals are keyboards, display units, and printer.

Example:

Monitor (*Visual Output Device*)	: CRT, LCD
Key Board (*Input Device*)	: light pen, mouse, touch screen, joy stick, digitizer
Printer (Hard Copy Device)	: Dot matrix (*impact*), thermal, ink jet, laser (*non-impact*)
Storage Device	: Magnetic tape, magnetic disk, optical disk

Disks are used mostly for bulk storage of programs and data. Magnetic disks have high speed rotational surfaces coated with magnetic material. Other input and output devices encountered in computer systems are digital incremental plotter, optical and magnetic character readers, analog to digital converters, and various data acquisition equipment.

Input and output devices that communicate with people and the computer are usually involved in the transfer of alphanumeric information to and from the device and the computer.

INPUT-OUTPUT INTERFACE

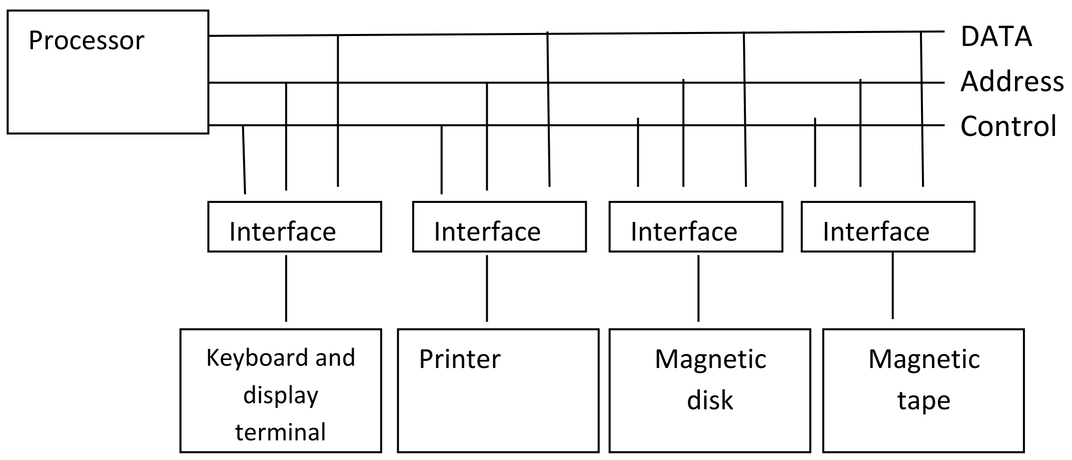
Input output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special **communication links** for interfacing them with the CPU.
The purpose of the **communication link** is to resolve the differences that exist between the **central computer** and each **peripheral**.
The major differences are:

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.

2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware components between the CPU and peripherals to **supervise and synchronize** all input and output transfers. These components are called **interface units** because they interface between the processor bus and the peripheral device.

In addition, each device may have its own controller that supervises the operations of the particular mechanism in the peripheral.



--Connection of I/O bus to input output devices--

ASYNCHRONOUS DATA TRANSFER

The internal operations in a digital system are **synchronized by means of clock pulses** supplied by a common **pulse generator**. Clock pulses are applied to all registers within a unit and all data transfers among internal registers occur simultaneously during the occurrence of a clock pulse.

Two units, such as a CPU and an I/O interface, are designed independently of each other.
Synchronous Data Transfer

- All data transfers occur simultaneously during the occurrence of a clock pulse
- Registers in the **interface** share a common clock with **CPU** registers

Asynchronous Data Transfer

- Internal timing in each unit (*CPU and Interface*) is independent
- Each unit uses its own private clock for internal registers
- This approach is widely used in most computer systems.

Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted.

One way of achieving this is by means of a strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.

Strobe control:

The strobe control method of asynchronous data transfer employs a **single control line** to time each transfer. The strobe may be activated by either the **source or the destination** unit.

The data bus carries the binary information from source unit to the destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available in the bus.

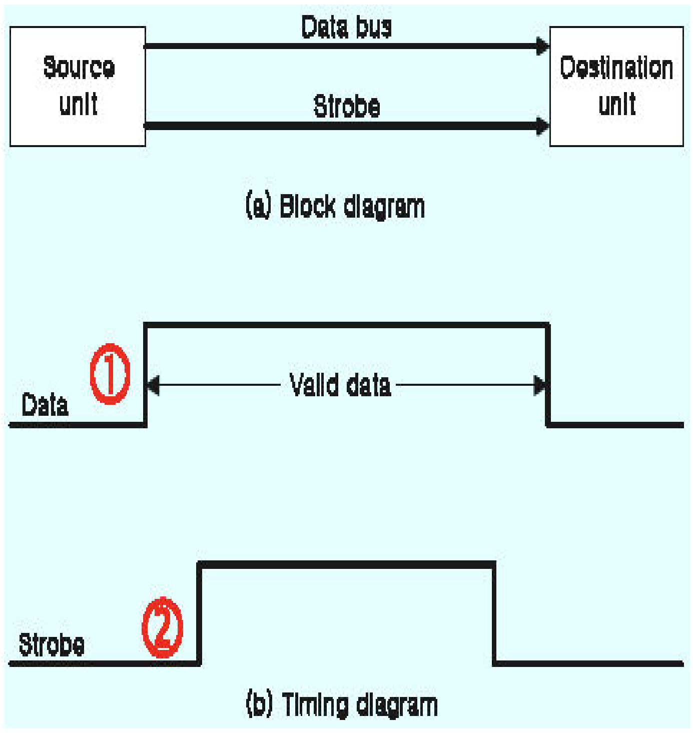


Fig. Source-initiated strobe

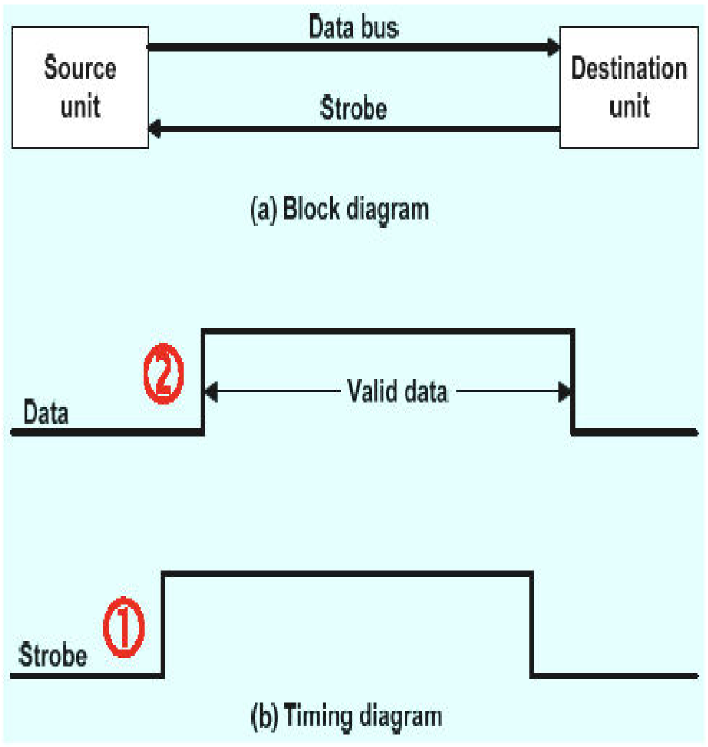


Fig. Destination-initiated strobe

As shown in the timing diagram of Figure, the source unit first places the data on the data bus. After a brief delay to ensure that the data settle to a steady value, the source activates the strobe pulse. The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data.

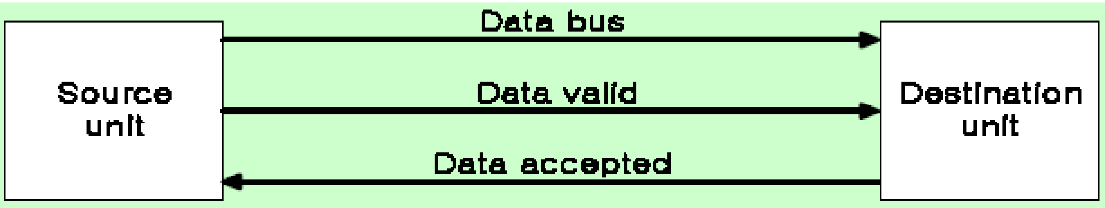
The source removes the data from the bus a brief period after it disables its strobe pulse. The fact that the strobe signal is disabled indicates that the data bus does not contain valid data. New valid data will be available only after the strobe is enabled again.

Figure shows a data transfer initiated by the destination unit. In the case the destination unit activates the strobe pulse, informing the source to provide the data. The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain in the bus long enough for the destination unit to accept it. The destination unit then disables the strobe. The source removes the data from the bus after a predetermined time interval.

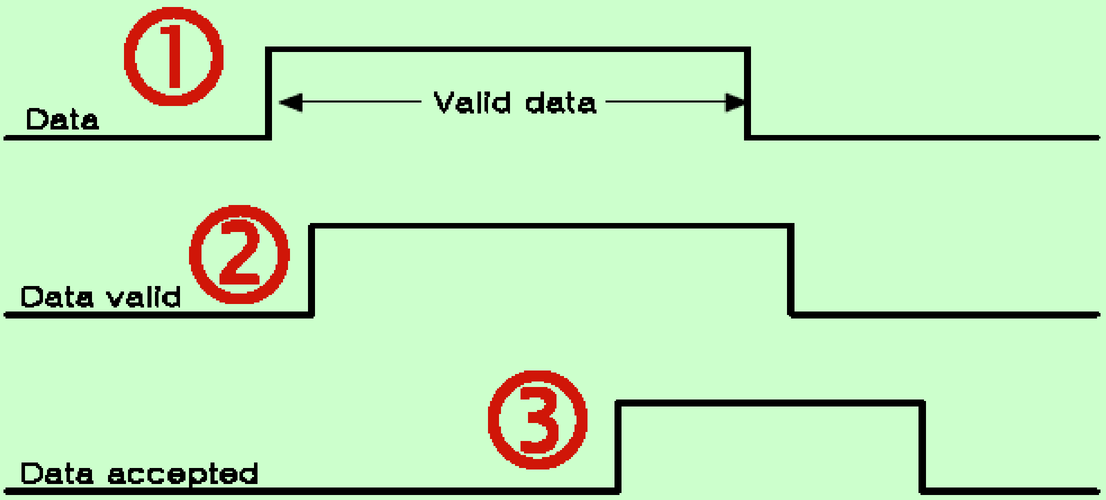
In many computers the strobe pulse is actually controlled by the clock pulses in the CPU. The CPU is always in control of the buses and informs the external units how to transfer data.

Handshaking

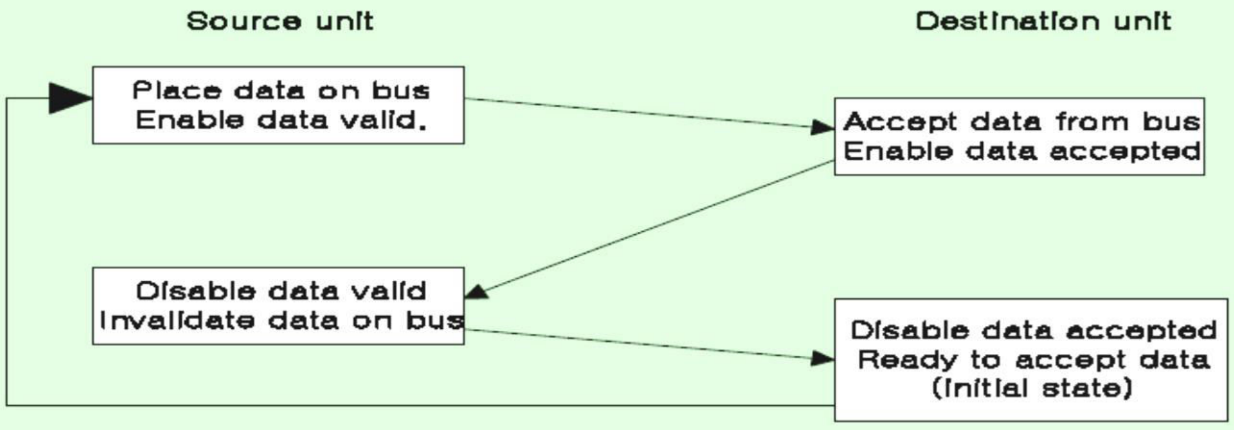
The disadvantages of the strobe method is that the source unit that indicates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus. Similarly, a destination unit that indicates the transfer has no way of knowing whether the source unit has actually placed the data on the bus.
The handshake method solves this problem by introducing a second control signal that provides a reply to the unit that initiates the transfer.
One control line is in the same direction as the data flow in the bus from the source to the destination. It is used by the source unit to inform the destination unit whether there are valid data in the bus. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept data. The sequence of control during the transfer depends on the unit that initiates the transfer.



(a) Block diagram



(b) Timing diagram



(c) Sequence of events

Asynchronous Serial Transfer

The transfer of data between two units may be done inparallel or serial.

- In parallel data transmission, each bit of the message has its own path and the total message is transmitted at the same time. This means that an n-bit message must be transmitted through n separate conductor paths.
  In serial data transmission, each bit in the message is sent in sequence one at a time. This method requires the use of one pair of conductors or one conductor and a common ground.
- Parallel transmission is faster but requires many wires. It is used for short distances and where speed is important.
- Serial transmission is slower but is less expensive since it requires only one pair of conductors.

MODES OF TRANSFER

Binary information received from an external device is usually stored in memory for later processing. Information transferred from the central computer into an external device originates in the memory unit.

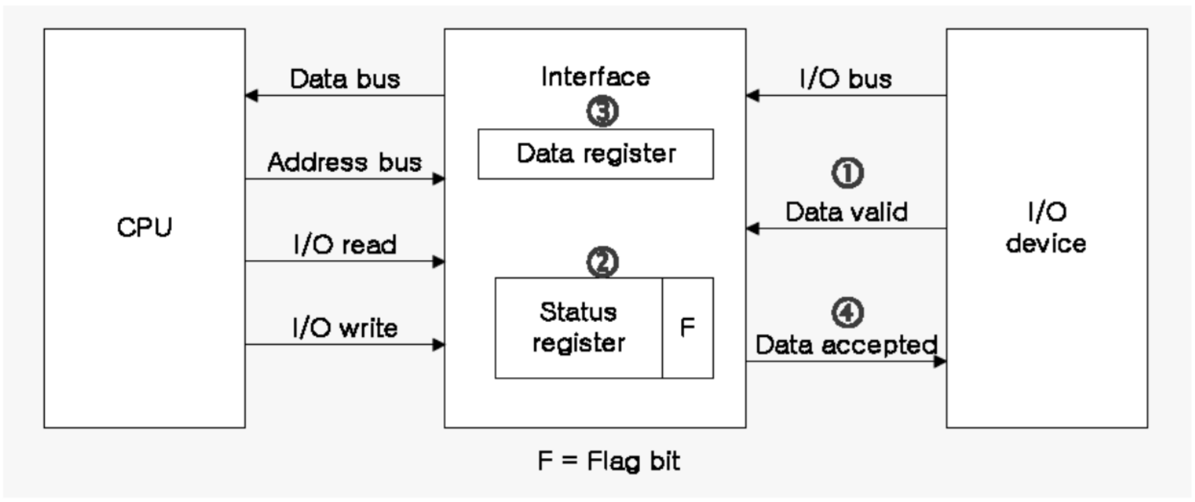
The CPU merely executes the I/O instructions and may accept the data temporarily, but the ultimate source or destination is the memory unit.

Data transfer between the central computer and I/O devices may be handled in a variety of modes. Some modes use the CPU as an intermediate path; others transfer the data directly to and from the memory unit.

Data transfer to and from peripherals may be handled in one of three possible modes:

1. Programmed I/O :

Programmed I/O operations are the result of I/O instructions written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually, the transfer is to and from a CPU register and peripheral. Other instructions are needed to transfer the data to and from CPU and memory. Transferring data under program control requires constant monitoring of the peripheral by the CPU.
In the programmed I/O method, the I/O device does not have direct access to memory. A transfer from an 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 a store instruction to transfer the data from the CPU to memory. Other instructions may be needed to verify that the data are available from the device and a count the number of words transferred.



Example of Programmed I/O:

An example of data transfer from an I/O device through an interface into the CPU is shown in the above figure. The device transfers bytes of data one at a time as they are available. When a byte of data is available, the device places it in the I/O bus and enables its data valid line. The interface accepts the byte into its data register and enables the data accepted line. The interface sets a bit in the status register that we will refer to as an F or "flag" bit.

The device can now disable the data valid line, but it will not transfer another byte until the data accepted line is disabled by the interface. This is according to the handshaking procedure established.

A program is written for the computer to check the flag in the status register to determine if a byte has been placed in the data register by the I/O device. This is done by reading the status register into a CPU register and checking the value of the flag bit. If the flag is equal to 1, the CPU reads the data from the data register. The flag bit is then cleared to 0 by either the CPU or the interface, depending on how the interface circuits are designed. Once the flag is cleared, the interface disables the data accepted line the device can then transfer the next data byte.

The programmed I/O method is particularly useful in small low speed computers. The difference in information transfer rate between the CPU and the I/O device makes this type of transfer inefficient.
Ie., the CPU is wasting time while checking the flag instead of doing some other useful processing tasks.

2) Interrupt-initiated I/O :

An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility.

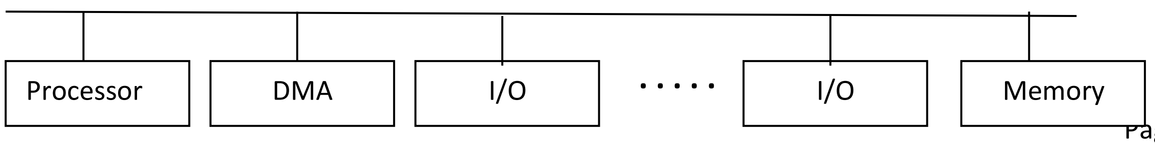
While the CPU is running a program, it does not check the flag. However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed that the flag has been set.

The CPU deviates from when it is doing to take care of the input or output transfer. After the transfer is completed, the computer returns to the previous program to continue what it was doing before the interrupt.

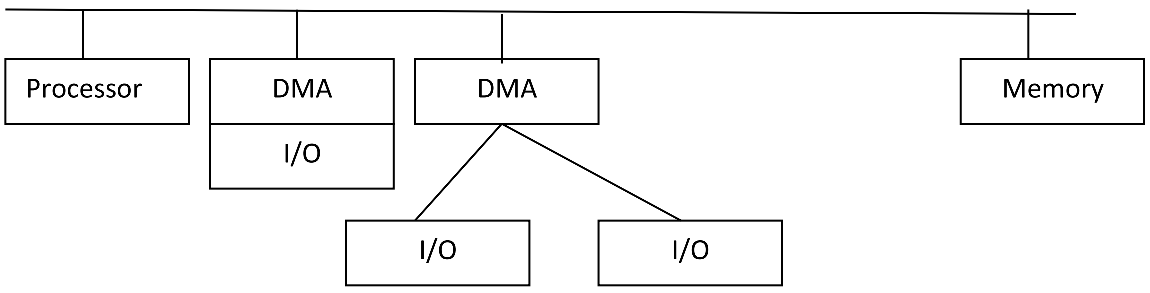
The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer. The way that the processor chooses the branch address of the service routine varies from one unit to another.

3. DIRECT MEMORY ACCESS (DMA):

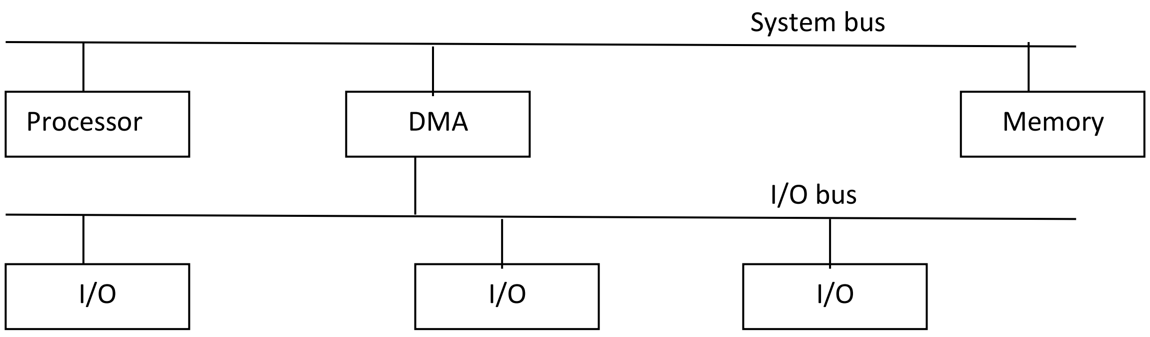
The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called direct memory access (DMA).



1)--Single bus, detached DMA--

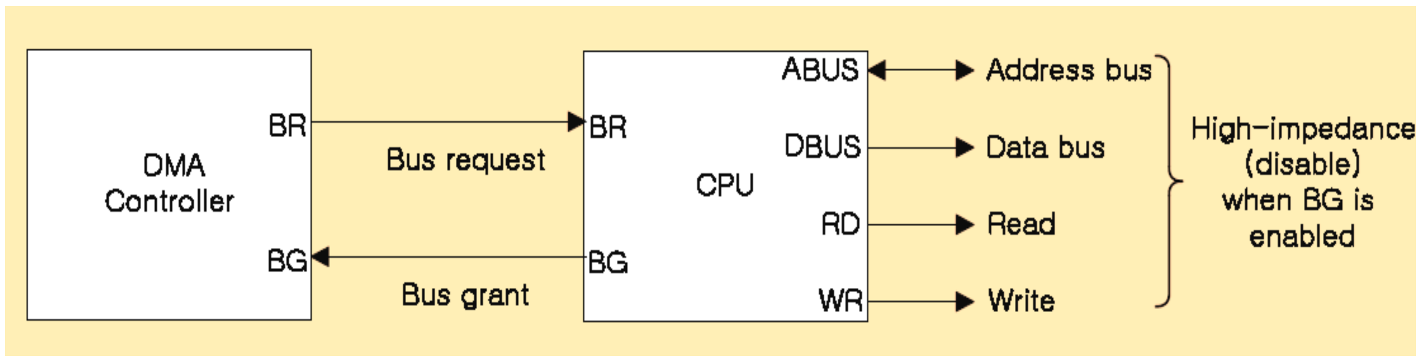


2)--Single bus, Integrated DMA-I/O--



3)--I/O Bus--

During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory. The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessors is to disable the buses through special control signals.



--CPU bus signals for DMA transfer--

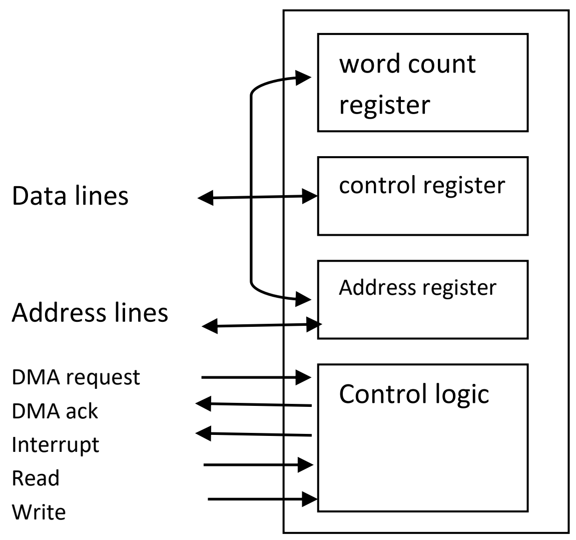
The figure shows two control signals in the CPU that facilitate the DMA transfer. The bus request (BR) input is used by the DMA controller to request the CPU to release control of the buses. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, the data bus, and the read and write lines into a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance.

The CPU activates the bus grant (BG) output to inform the external DMA that the buses are in the high impedance state. The DMA that originated the bus request can now take control of the buses to conduct memory transfers without processor intervention. When the DMA terminates the transfer it disables the bus request line.

The CPU disables the bus grant, takes control of the buses, and returns to its normal operation. When the DMA takes control of the bus system, a communicates directly with the memory. An alternative technique called cycle stealing allows the DMA controller to transfer one data word at a time, after which a must return control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle.

When the transfer is complete, the DMA module sends an interrupt signal to the processor. Thus, the processor is involved only at the beginning and end of the transfer.

Block diagram of DMA controller:



The DMA is first initialized by the CPU. After that, the DMA starts and continues to transfer data between memory and peripheral unit until an entire block is transferred.

When the CPU wishes to read or write a block of data, it issues a command to the DMA module, by sending to the DMA module the following information through the data bus:

- The starting location in memory to read from or write to, communicated on the data lines
- The number of words to be read or written, again communicated via the data lines and stored in the stored count register.
- Control to specify the mode of transfer such as read or write
- A control to start the DMA transfer.

The starting address is stored in the address register. The word count is stored in the word count register. Once the DMA is initialized, the CPU stops communicating with the DMA and continue with other work unless it receives an interrupt signal or if it wants to check how many words have been transferred.

DMA transfer is very useful in many applications. It is used for fast transfer of information between magnetic disks and memory.

CHAPTER FIVE

MEMORY ORGANIZATION

4.1 MEMORY HIERARCHY

The hierarchical arrangement of storage in current computer architectures is called the memory hierarchy. The memory unit is an essential component in any digital computer since it is needed for storing programs and data.

- The memory unit that communicates directly with the CPU is called the Main Memory (or Primary memory).
- Devices that provide backup storage are called auxiliary Memory (or Secondary).

Only programs and data currently needed by the processor reside in Main memory. All other information is stored in Auxiliary memory and transferred to main memory when needed.

The Memory hierarchy system consists of all storage devices employed in a computer system from the slow but high capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high speed processing logic.

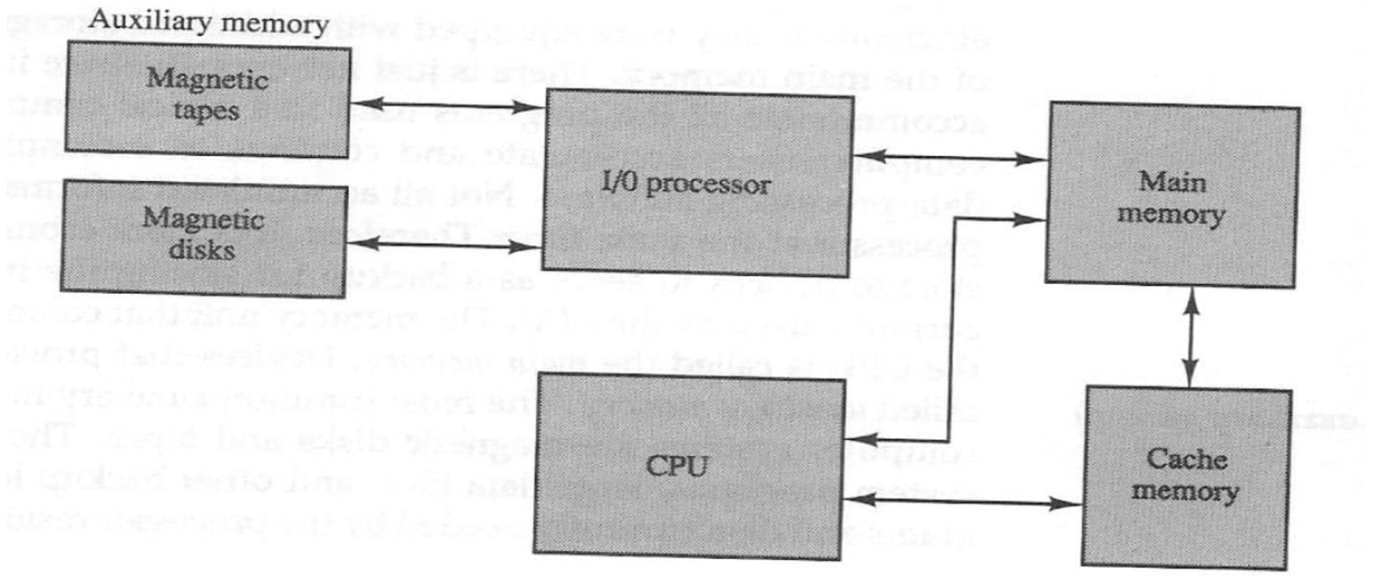


Figure: 5.1 --Memory Hierarchy in a computer system--

When programs not residing in main memory are needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data.

The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic.

The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.

By making programs and data available at a rapid rate, it is possible to increase the performance rate of the computer. While the I/O processor manages data transfers between auxiliary memory and main memory, the cache organization is concerned with the transfer of information between main memory and CPU. Thus each is involved with a different level in the memory hierarchy is economics.

4.2 MAIN MEMORY

It is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data during the computer operation. The principal technology used for the main memory is based on semiconductor integrated circuits. Integrated circuit RAM chips are available in two possible operating modes, static and dynamic.

The static RAM (SRAM) consists essentially of internal flip flops that store the binary information. The stored information remains valid as long as power is applied to the unit.

The dynamic RAM (DRAM) stores the binary information in the form of electric charges that are applied to capacitors. The stored charge on the capacitors tend to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory. Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write cycles.

The ROM portion of main memory is needed for storing an initial program called a **Bootstrap Loader**. The bootstrap loader is a program whose function is to start the computer software operating when power is turned on. The contents of ROM remain unchanged after power is turned off and on again. The start-up of a computer consists of turning the power on and starting the execution of an initial program. Thus when power is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader. The bootstrap program loads a portion of the operating system from disk to main memory and control is then transferred to the operating system, which prepares the computer for general use.

RAM and ROM chips are available in a variety of sizes. If the memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to form the required memory size.

4.3 AUXILIARY MEMORY

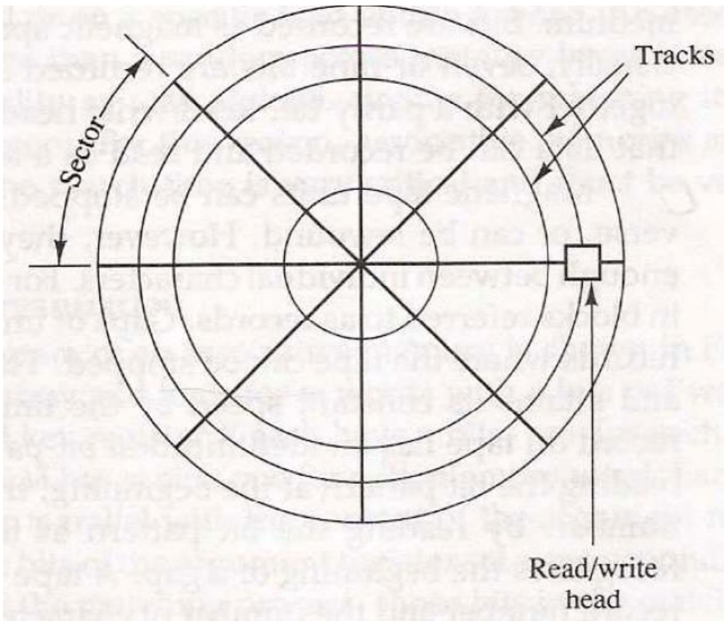
The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. The important characteristics of any device are its access mode, access time, transfer rate and capacity. The average time required to reach a storage location in memory and obtain its contents is called the **access time**.

The access time = seek time + transfer time

- Seek time : required to position the read-write head to a location

- Transfer time : required to transfer data to or from the device

Auxiliary storage is organized in records or blocks. A record is a specified number of characters or words. Reading or writing is always done on entire records. Magnetic drums and disks are consisting of high speed rotating surfaces coated with a magnetic recording medium. The recording surface rates at uniform speed. Bits are recorded as magnetic spots on the surface as it passes a stationary mechanism called a write head.



Stored bits are detected by a change in magnetic field produced by a recorded spot on the surface as it passes through a read head.

Magnetic disks:

A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface. All disks rotate together at high speed and are not stopped or started for access purposes. Bits are stored in the magnetized surface in spots along concentric circles called tracks. The tracks are commonly divided into sections called sectors.

Tracks and sectors

a) All tracks have same number of sectors
b) Outer tracks have more sectors than inner tracks.

The number of bytes stored in each sector is kept same. All tracks store the same amount of data. This results higher bit density in inner tracks than that of the outer tracks.

Since the same number of bytes is stored in each sector, the size of the inner sectors decides the storage capacity for all other sectors on the disk. Ex. Hard disk, floppy disk...

Optical disks:

Optical disks are used for backup memory. Information is written to or read from an optical disk using laser beam. It has very high storing capacity as compared to magnetic floppy disks. It has very long life. The capacity of optical disks varies from 650 MB to 17 GB. DVDs of 15,25,30 and 50 GB capacity etc., an optical disk is a direct access device. As its read /write head does not touch the disk surface, there is no disk wear and problem of head crash. Elaborate error checking codes can be used as there is no problem of space because of its high storage capacity.

The greatest drawback of an optical disk drive system is its large access time as compared to magnetic hard disk drive. An optical disk system the drive has to move on a sizable optical assembly across the disk surface. This results in an increased access time.

Types of optical disks: CD(compact disk), CD-R(recordable), CD-RW(read/write), DVD(digital versatile disk), DVD-R, DVD-RW..

4.4CACHE MEMORY

If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced,Thus reducing the total execution time of the program. Such a fast small memory is referred to as cache memory.

- The cache is the fastest component in the memory hierarchy and approaches the speed of CPU component.

  When CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.

The performance of cache memory is frequently measured in terms of a quantity called **hit ratio.** When the CPU refers to memory and finds the word in cache, it is said to produce a **hit.** Otherwise, it is a **miss.**

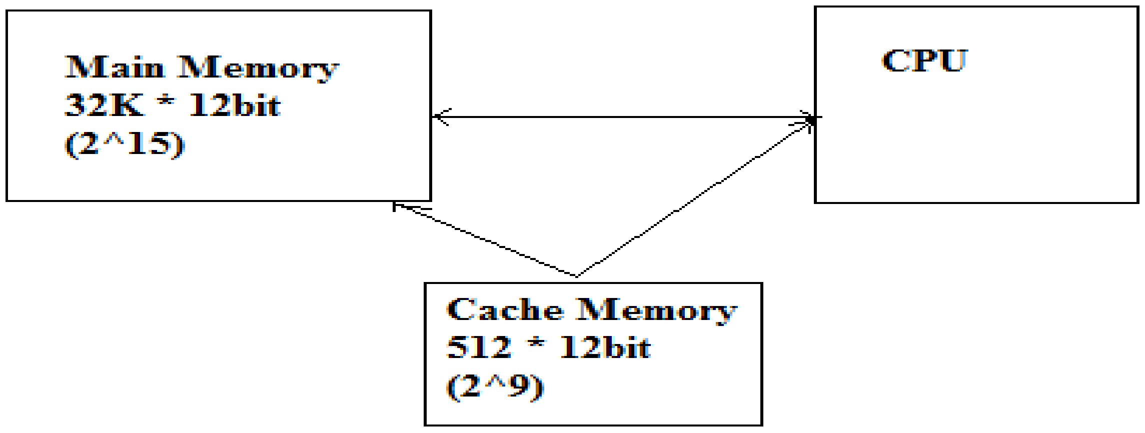
Hit ratio = hit / (hit+miss)

- The basic characteristic of cache memory is its fast access time,
- Therefore, very little or no time must be wasted when searching the words in the cache

The transformation of data from main memory to cache memory is referred to as a **mapping** process, there are three types of mapping:

- Associative mapping
- Direct mapping
- Set-associative mapping

- To help understand the mapping procedure, we have the following example:



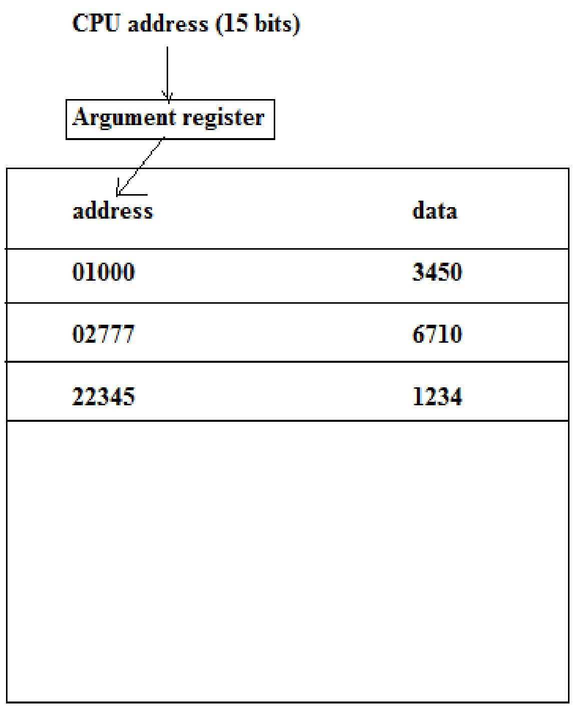
Associative mapping

- The fastest and most flexible cache organization uses an associative memory.
- The associative memory stores both the address and data of the memory word.
- This permits any location in cache to store any word from main memory.

- A main memory block can load into **any** line of cache
- The Memory Address is interpreted as **tag and word**
- The **Tag** uniquely identifies block of memory
- Every **line's tag** is examined for a match
- Cache searching gets expensive/complex or slow

The address value of 15 bits is shown as a five-digit **octal** number and its corresponding 12-bitword is shown as a four-digit octal number.

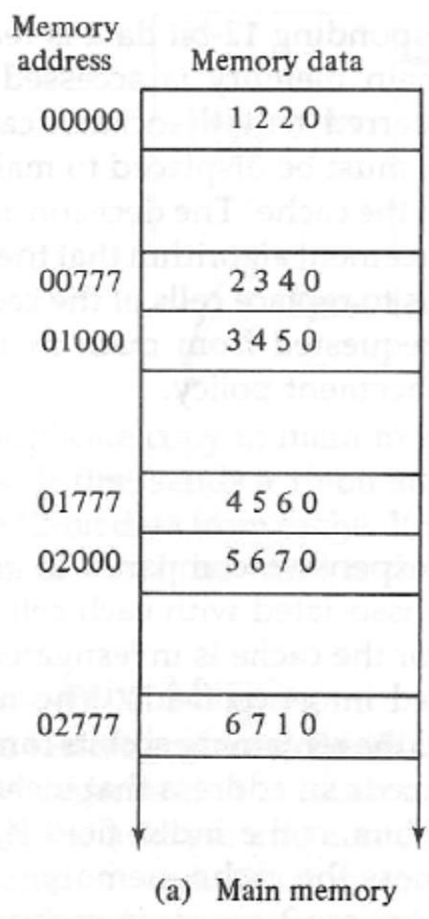
- A CPU address of 15 bits is places in the argument register and the associative memory searched for a matching address.
- If the address is found, the corresponding 12-bits data is read and sent to the CPU.If not, the main memory is accessed for the word.
- If the cache is full, an address-data pair must be displaced to make room for a pair that is needed and not presently in the cache.



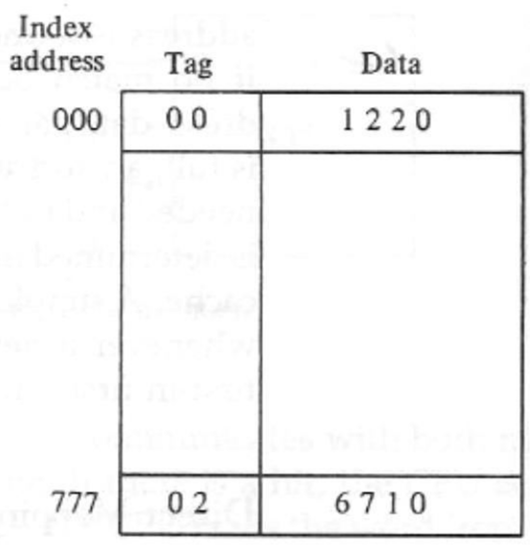
address	data
01000	3450
02777	6710
22345	1234

Direct Mapping

- Associative memory is expensive compared to RAM.
- In general case, there are 2^k words in cache memory and 2^n words in main memory (in our case, k=9, n=15)
- The n bit memory address is divided into two fields: k-bits for the index and n-k bits for the tag field
- Each block of main memory maps to only one cache line
- i.e. if a block is in cache, it must be in one specific place
- Address is in two parts
- - Least Significant w bits identify unique word
- - Most Significant s bits specify which one memory block



(a) Main memory



(b) Cache memory

The internal organization of the words in the cache memory is as shown in the above figure. Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the **index** field is used for the address to access the cache.

The tag field of the CPU address is compared with the tag in the word read from the cache. If the two **tags** match, there is a hit and the desired data word is in cache. If there is no match, there is a miss and the required word is read from main memory. It is then the cache together with the new tag, replacing the previous value.

The disadvantage of direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly.

Set-Associative Mapping

- The disadvantage of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time.
- Cache is divided into a number of **sets**
- Each set contains a **number of lines**
- A given block maps to any line in a given set; e.g. Block B can be in any line of set ***i***

- Set-Associative Mapping is an improvement over the direct-mapping in that each word of cache can store two or more word of memory under the same index address.

4.5VIRTUAL MEMORY

Virtual memory is used to give programmers the illusion that they have a very large memory at their disposal, even though the computer actually has a relatively small main memory.

A virtual memory system provides a mechanism for translating program generated addresses into correct main memory locations. This is done dynamically, while programs are being executed in the CPU. The translation or mapping is handled automatically by the hardware by means of a mapping table.

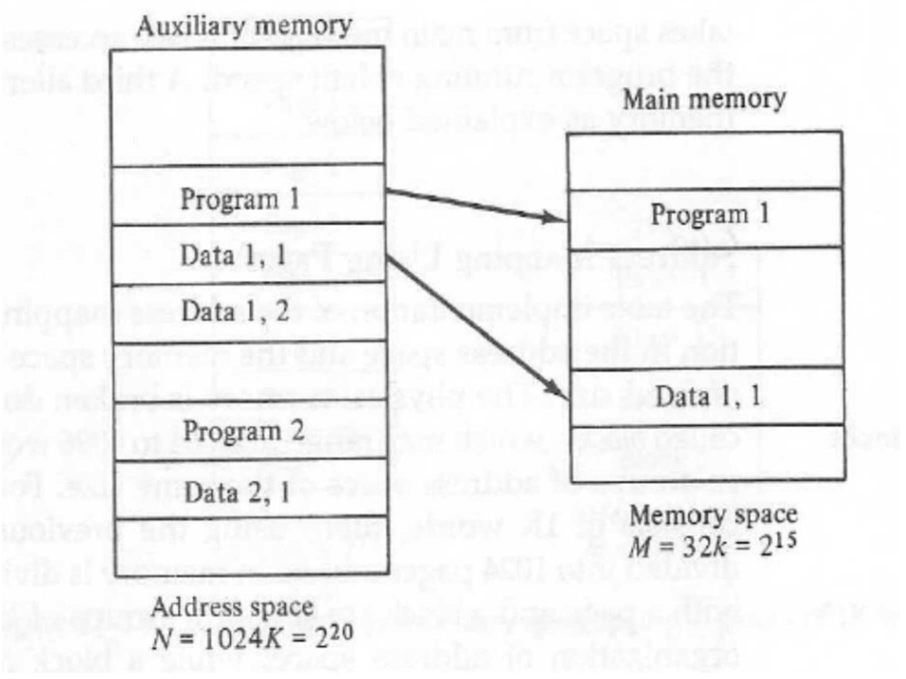


Figure: Relation between address and memory space in a virtual memory system

- The address used by a programmer will be called a virtual address or logical address.
- An address in main memory is called a physical address.

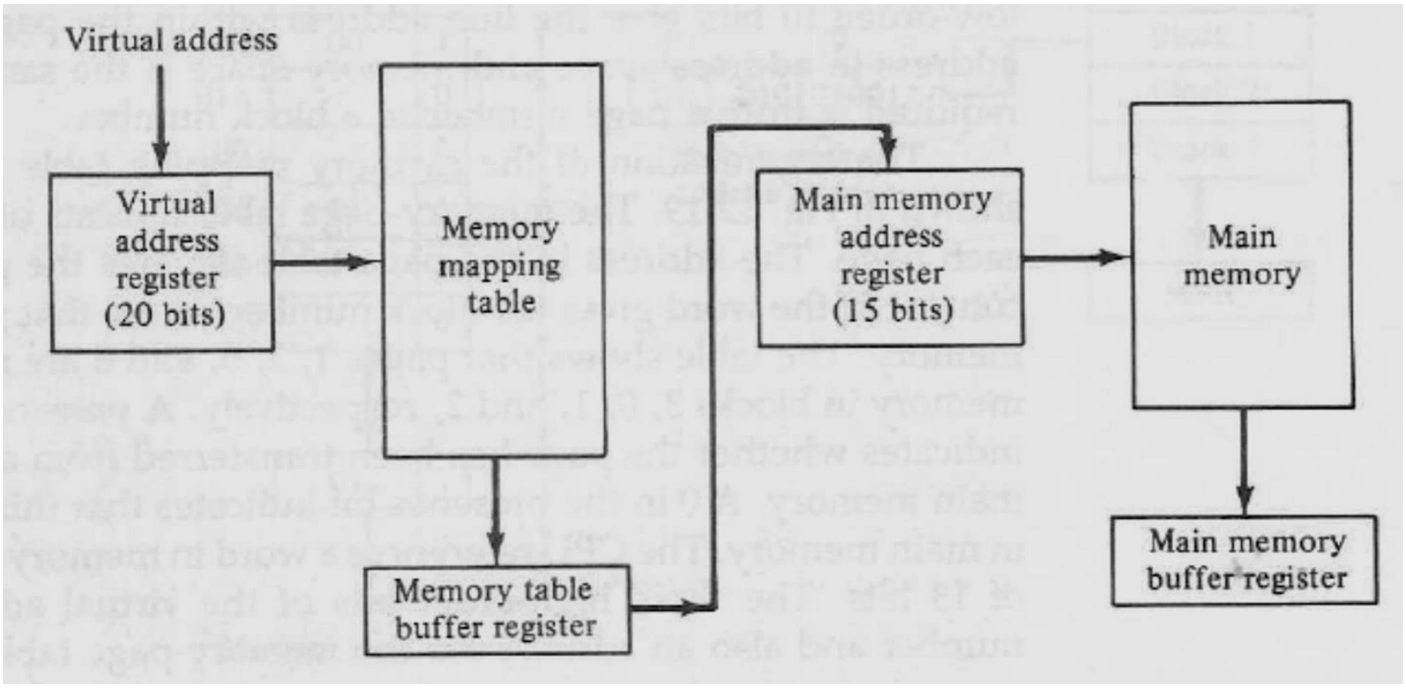

Virtual address
Virtual address register (20 bits)
Memory mapping table
Memory table buffer register
Main memory address register (15 bits)
Main memory
Main memory buffer register

Figure: Memory table for mapping a virtual address

Address mapping using pages: The table implementation of the address mapping is simplified if the information in the address space and the memory space are each divided into groups of fixed size. The term **page** refers to groups of address space of the same size.

For example, if a page or block consists of 1K words, then, using the previous example, address space is divided into 1024 pages and main memory is divided into 32 blocks. Although both a page and a block are split into groups of 1K words, a page refers to the organization of address space while a block refers to the organization of memory space. The programs are also considered to be split into pages.

Portions of programs are moved from auxiliary memory to main memory in records equal to the size of a page. The term "page frame" is sometimes used to denote a block.

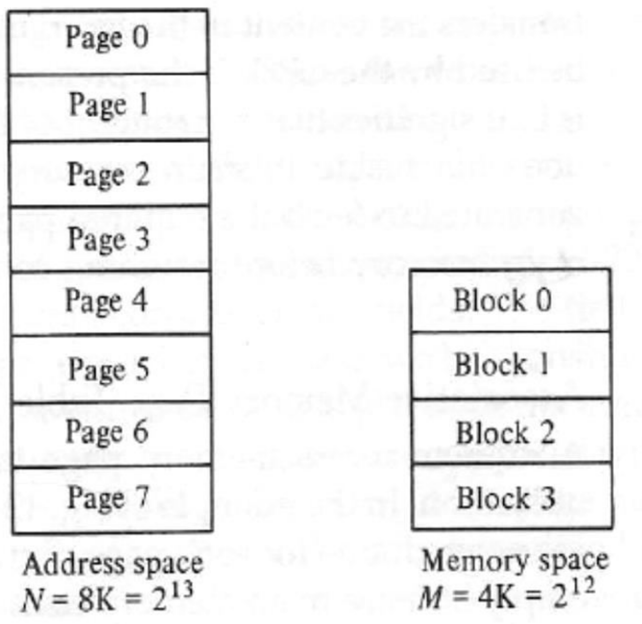


Figure: Address space and memory space split into groups of 1K words

Consider a computer with an address space of 8K and a memory space of 4K. If we split each into groups of 1K words we obtain eight pages and 4 blocks. At any given time, upto four pages of address space may reside in main memory in any one of the four blocks. The mapping from address space to memory space is facilitated if each virtual address is considered to be represented by two numbers: a page number address and a line within the page.

Replacement policies:

A virtual memory system is a combination of hardware and software technique. The memory management software system handles all the software operations for the efficient utilization of memory space. It must decide

(1) Which page in main memory ought to be removed to make room for a new page

(2) When a new page is to be transferred from auxiliary memory to main memory, and

(3) Where the page is to be placed in main memory.

When a program starts execution, one or more pages are transferred into main memory and the page table is set to indicate their position. The program is executed from main memory until it attempts to reference a page that is still in auxiliary memory. This condition is called page fault. When page fault occurs, the execution of the present program is suspended until the required page is brought into main memory. Since loading a page from auxiliary memory to main memory is basically an I/O operation, the operating system assigns this task to the I/O processor. In the meantime, control is transferred to the next program in memory that is waiting to be processed in the CPU. Later, when the memory block has been assigned and the transfer completed, the original program can resume its operation.

When a page fault occurs in a virtual memory system, it signifies that the page referenced by the CPU is not in main memory. A new page is then transferred from auxiliary memory to main memory. If main memory is full, it would be necessary to remove a page from a memory block to make room for the new page. The policy for choosing pages to remove is determined from the replacement algorithm that is used. Two of the most common replacement algorithms used are the first-in, first-out (FIFO) and the least recently used (LRU). The FIFO algorithm selects for replacement the page that has been in memory the longest time. Each time a page is loaded into memory, its identification number is pushed into a FIFO stack.

FIFO will be full whenever memory has no more empty blocks. When a new page must be loaded, the page least recently brought in is removed. The page to be removed is easily determined because its identification number is at the top of the FIFO stack. The FIFO replacement policy has the advantage of being easy to implement. It has the disadvantage that under certain circumstances pages are removed and loaded from memory too frequently.

The LRU policy is more difficult to implement but has been more attractive on the assumption that the least recently used page is a better candidate for removal than the least recently loaded page as in FIFO. The LRU algorithm can be implemented by associating a counter with every page that is in main memory. At fixed intervals of time, the counters associated with all pages presently in memory are incremented by 1. The least recently used page is the page with the highest count. The counters are often called aging registers, as their count indicates their age, that is, how long ago their associated pages have been referenced.

Memory management related hardware: A memory management system is a collection of hardware and software procedures for managing the various programs residing in memory. The memory management software is part of an overall operating system available in many computers.

Here we are concerned with the hardware unit associated with the memory management system. The basic components of a memory management unit are:

1. A facility for dynamic storage relocation that maps logical memory references into physical memory addresses.
2. A provision for sharing common programs stored in memory by different users.
3. Protection of information against unauthorized access between users and preventing users from changing operating system functions.

The fixed page size used in the virtual memory system causes certain difficulties with respect to program size the logical structure of programs. It is more convenient to divide programs and data into logical parts called segments.

A segment is a set of logically related instructions or data elements associated with a given name.

Segments may be generated by the programmer or by the operating system. Examples of segments are a subroutine, an array of data, a table of symbols, or a user's program. The sharing of common programs is an integral part of a multiprogramming system. For example, several users wishing to compile their C programs should be able to share a single copy of the compiler rather than each user having a separate copy in memory. Other system programs residing in memory are also shared by all users in a multiprogramming system without having to produce multiple copies.

The function of the memory management unit is to map logical addresses into physical addresses similar to the virtual memory mapping concept.

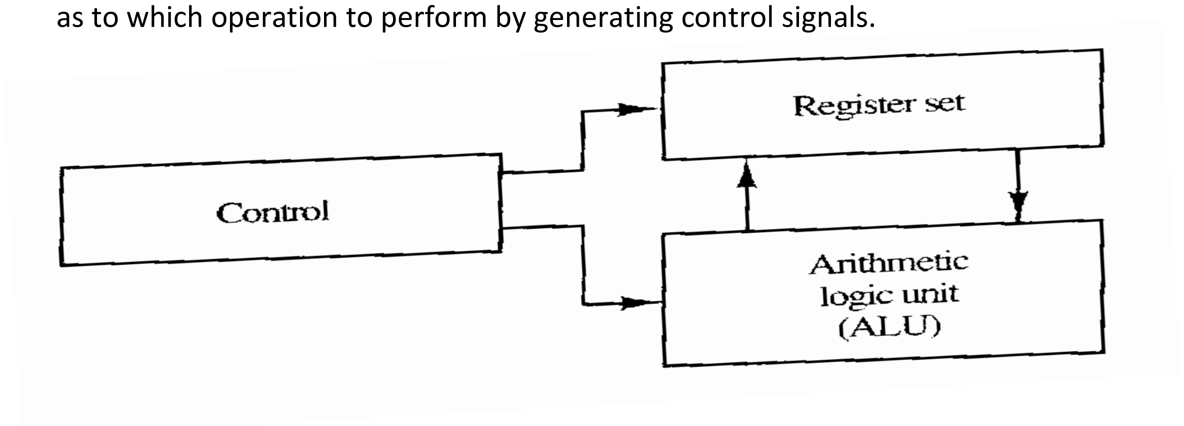
CHAPTER FIVE

CENTRAL PROCESSING UNIT (CPU)

- The part of the computer performs the bulk of data processing operations is called the central processing unit
- The CPU is made up of three major parts:
  - Register set
  - ALU
  - Control units

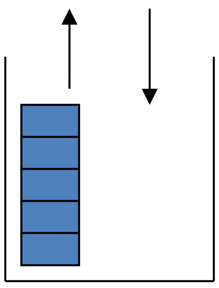
The **central processing unit (CPU)** of a computer is the main unit that dictates the rest of the computer organization.

- 1. **Register se**t: Stores intermediate data during the execution of instructions;
- 2. **Arithmetic logic unit (ALU**): Performs the required micro-operations for executing the instructions;
- 3. **Control uni**t: supervises the transfer of information among the registers and instructs the ALU as to which operation to perform by generating control signals.



STACK ORGANIZATIONS

- **Stac**k: A storage device that stores information in such a manner that the item stored last is the first item retrieved.
- Also called **last-in first-out (LIFO) lis**t. Useful for compound arithmetic operations and nested subroutine calls.



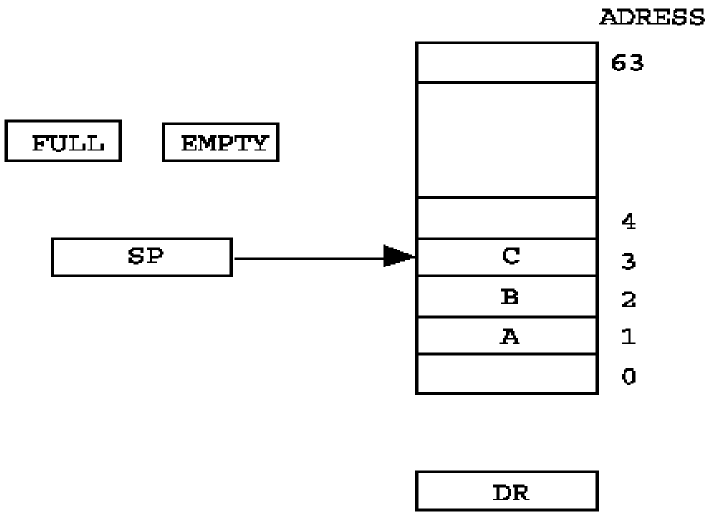
- ***Stack pointer (SP****): A register that holds the address of the top item in the stack.*

  ***SP** always points at the top item in the stack*
- ***Push**: Operation to insert an item into the stack.*
- ***Pop**: Operation to retrieve an item from the stack.*

REGISTER STACK

• A stack can be organized as a collection of a finite number of registers.

For example, consider the organization of a 64 word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack



Three items are placed in the stack: A, B and C. Item C is on top of the stack so that the content of SP is now 3. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2.

To inset a new item, the stack is pushed by incrementing SP and writing a word in the next higher location of the stack. Note that item C has been read out but not physically removed. This does not matter because when the stack is pushed, a new item is written in its place. In a 64 word stack, the stack pointer contains 6 bits because 2^6=64. Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 is incremented by 1, the result is 0 since 111111 +1=1000000 in binary, but SP can accommodate only the six least significant bits. Similarly, when 000000 is decremented by 1, the result is 111111.

- The one bit register FULL is set to 1 when the stack is full, and one bit register EMTY is set to 1 when the stack is empty of items.
- DR is the data register that holds the binary data to be written into or read out of the stack.
- Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If stack is not full (if FULL=0), a new item is inserted with a push operation.

The push operation is implemented with the following sequence of Microoperations:

The following are the micro-operations associated with the stack

Initialization
SP ¬ 0, EMPTY ¬ 1, FULL ¬ 0

Push

SP ←SP+1	increment stack pointer
M[SP] ←DR	write item on top of the stack
If(SP=0) then (FULL ←1)	check if stack is full
EMTY ←0	Mark the stack not empty

The stack pointer is incremented so that it points to the address of the next higher word. A memory write operation inserts the word from DR into the top of the stack. Note that SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP. The first item stored in the stack is at address 1. The last item is stored at address 0, if SP reaches 0, the stack is full of items, so FULL is set to 1.

Pop

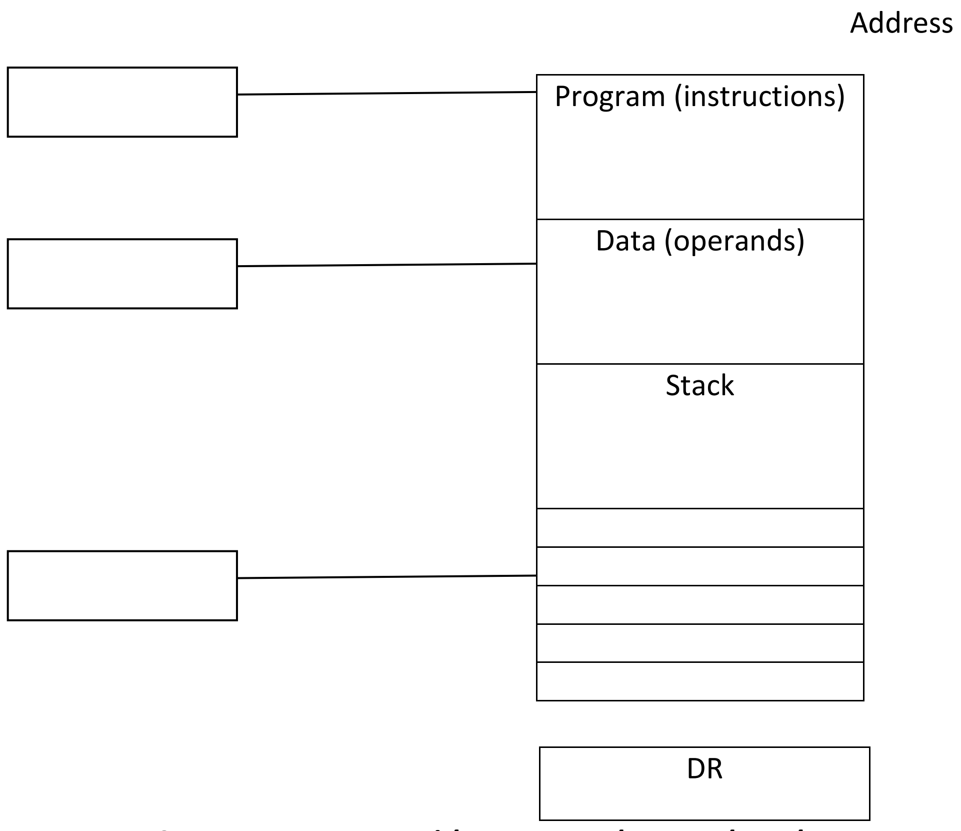
DR ← M[SP]	Read item from the top of stack
SP ← SP-1	Decrement stack pointer
If (SP=0) then (EMTY ← 1)	check if stack is empty
FULL ←0	Mark the stack not full

The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1. This condition is reached if the item read was in location 1. Once the item is read out, SP is decremented and reaches the value 0, which is the initial value of SP.

<u>Memory Stack:</u>

The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. A portion of computer memory partitioned into three segments: program, data and stack.

The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data. the stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or from the stack.



--Computer memory with Program, data, and stack segments--

The initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack is 3000. No provisions are available for stack limit checks.

A new item is inserted with the push operation as follows:

$SP \leftarrow SP-1$

$M[SP] \leftarrow DR$

The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack.

A new item is deleted with a pop operation as follows:

$DR \leftarrow M[SP]$

$SP \leftarrow SP+1$

The top item is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack. Most computers do not provide hardware to check for stack overflow. The stack limits can be checked by using two processor registers: one to hold the upper limit (3000), and the other to hold the lower limit (4001). After a push operation, SP is compared with the upper limit register and after a pop operation, SP is compared with the lower limit register.

The two microoperations needed for either the push or pop are (1) an access to memory through SP, and (2) updating SP. The advantage of a memory stack is that the CPU can refer to it without having to specify an address, since the address is always available and automatically updated in the stack pointer.

REVERSE POLISH NOTATION (postfix)

A stack organization is very effective for evaluating arithmetic expressions. The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer. The common arithmetic expressions are written in infix notation, with each operator written between the operands.

Consider the simple arithmetic expression.
A*B + C*D
To evaluate this arithmetic expression it is necessary to compute the product A*B, store the product while computing C*D, and then sum the two products.

The polish mathematician Lukasiewicz showed that arithmetic expressions can be represented in prefix notation. The postfix notation, referred to as Reverse Polish Notation (RPN), places the operator after the operands.

A+B	infix notation
+AB	Prefix or Polish notation
AB+	Postfix or Reverse Polish Notation

The reverse polish notation is in a form suitable for stack manipulation. The expression

A*B+C*D is written in reverse polish notation as

AB*CD*+

Scan the expression from left to right. When an operator is reached, perform the operation with the two operands found on the left side of the operator. Continue to scan the expression and repeat the procedure for every operator encountered until there are no more operators.

We perform the operation A*B and replace A,B and * by the product to obtain

(A*B)CD*+

Where (A*B) is a single quantity obtained from the product. The next operator is a * and its previous two operands are C and D, so we perform C *D and obtain an expression with two operands and one operator:

(A*B)(C*D)+ the next operator is + and the two operands to be added are the two products, so we add the two quantities to obtain result.

This hierarchy dictates that we first perform all arithmetic inside inner parentheses, then inside outer parentheses, and do multiplication and division operation before addition and subtraction operations. (ie., BODMAS rule → Bracket of division, multiplication, addition and subtraction)

Consider the expression: (A+B)*[C*(D+E)+F], The converted expression is AB+DE+C*F+*. Proceeding from left to right, we first add A and B, then add D and E. At this point we are left with (A+B)(D+E)C*F+*

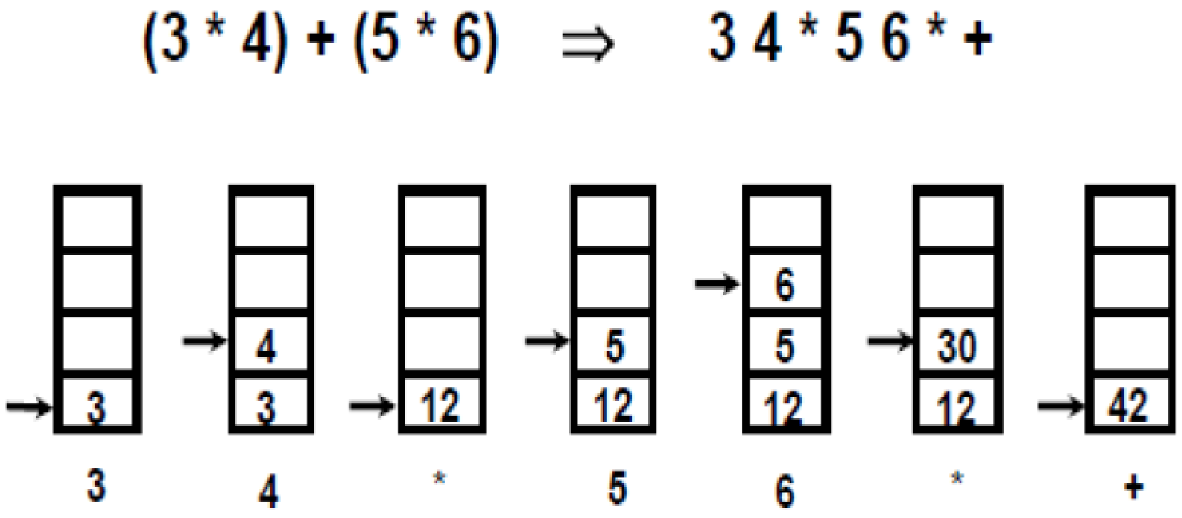
<u>Evaluation of Arithmetic Expressions:</u>

The following microoperations are executed with the stack when an operation is entered in a calculator or issued by the control in a computer.

1. The two topmost operands in stack are used for the operation, and
2. The stack is popped and the result of the operation replaces the lower operand

The following numerical example may clarify this procedure. Consider the arithmetic expression. (3*4) + (5*6)

In reverse polish notation, this is expressed as 34*56*+



-Stack operations --

Each box represents one stack operations and the arrow always points to the top of the stack. Scanning the expression from left to right, we encounter 2 operands. First the number 3 is pushed into the stack, then the number 4. The next symbol is the multiplication operator *. This causes a multiplication of the two topmost items in the stack. The stack is then popped and the product on top of the stack, replacing the two original operands. Next we encounter the two operands 5 and 6 so they are pushed into the stack. The stack operation that results from the next * replaces these two numbers by their product. The last operation causes an arithmetic addition of the two topmost numbers in the stack to produce the final result of 42.

Most compliers, irrespective of their CPU organization, convert all arithmetic expressions into Polish notation anyway because this is the most efficient method for translating arithmetic expressions into machine language instructions.

Most computers fall into one of 3 types of CPU organizations:

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

INSTRUCTION FORMATS

A computer will usually have a variety of instruction code formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.

The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

1. **Mode fiel**d: Specifies the way the effective address is determined
2. **Operation cod**e: Specifies the operations to be performed.
3. **Address fiel**d: Designates a memory address or a processor register

Mode	Opcode	**Address**

Instruction Format

- **Zero address instruction: Stack** is used. Arithmetic operation pops two operands from the stack and pushes the result.
- **One address instructions: AC (Accumulator Register)** and **memory**. Since the accumulator always provides **one operand**, only **one memory address** needs to be specified.
- **Two address instructions: Two address registers** or **two memory locations** are specified, one for the final result.
- **Three address instructions:** Three address registers or memory locations are specified, one for the final result. It is also called general address organization

❖ *To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement*

X = (A + B) * (C + D)

We will use the symbols ADD, SUB, MUL and DIV for the 4 arithmetic operations.
MOV for the transfer type operation
LOAD and STORE for transfers to and from memory and AC register.

We will assume that the operands are in memory addresses A,B,C, and D, and the result must be stored in memory at address X

Three Address Instructions:

```
ADD    R1, A, B              ie., R1 ← M[A] + M[B]
ADD    R2, C, D              i.e., R2 ← M[C] + M[D]
MUL    X, R1, R2             i.e., M[X] ← R1 * R2
```

It is assumed that the computer has two processor registers, R1, and R2. The symbol M[A] denotes the operand at memory address symbolized by A.

Two Address Instructions:

```
MOV    R1, A         ie., R1 ← M[A]
ADD    R1, B         i.e., R1 ← R1 + M[B]
MOV    R2, C         ie., R2 ← M[C]
ADD    R2, D         i.e., R2 ← R2 + M[D]
MUL    R1, R2        i.e., R1 ← R1 * R2
MOV    X, R1         i.e., M[X] ← R1
```

One Address Instructions:

```
LOAD    A            ie., AC ← M[A]
ADD     B            i.e., AC ← AC + M[B]
STORE  T             i.e., M[T] ← AC
LOAD   C             ie., AC ← M[C]
ADD    D             i.e., AC ← AC + M[D]
MUL      T           i.e., AC ← AC * M[T]
STORE  X             i.e., M[X] ← AC
```

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

Zero Address Instructions:

A stack organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.
(*TOS stands for top of stack*)

```
PUSH   A      i.e., TOS ← A
PUSH   B      i.e., TOS ← B
ADD           i.e., TOS ← ( A + B )
PUSH   C      i.e., TOS ← C
PUCH   D      i.e., TOS ← D
ADD           i.e., TOS ← ( C + D )
MUL           i.e., TOS ← ( C + D ) * ( A + B )
POP    X      i.e., M[X] ← TOS
```

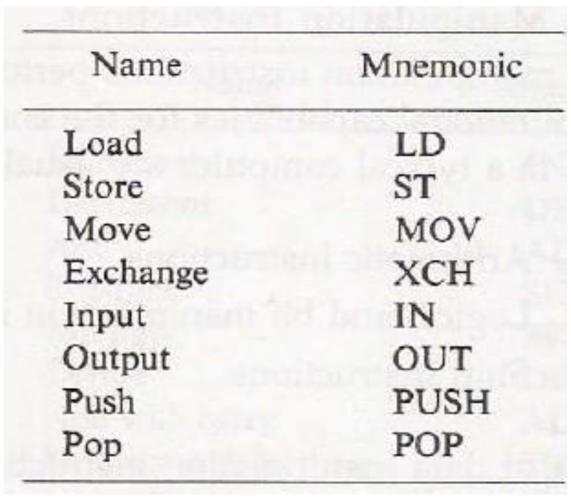
DATA TRANSFER AND MANIPULATION

Most computer instructions can be classified into three categories:

1) Data transfer, 2) Data manipulation, 3) Program control instructions

Data Transfer Instruction

Typical Data Transfer Instruction :



Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Load : transfer from memory to a processor register, usually an AC (*memory read*)

Store : transfer from a processor register into memory (*memory write*)

Move : transfer from one register to another register

Exchange : swap information between two registers or a register and a memory word

Input/Output : transfer data among processor registers and input/output device

Push/Pop : transfer data between processor registers and a memory stack