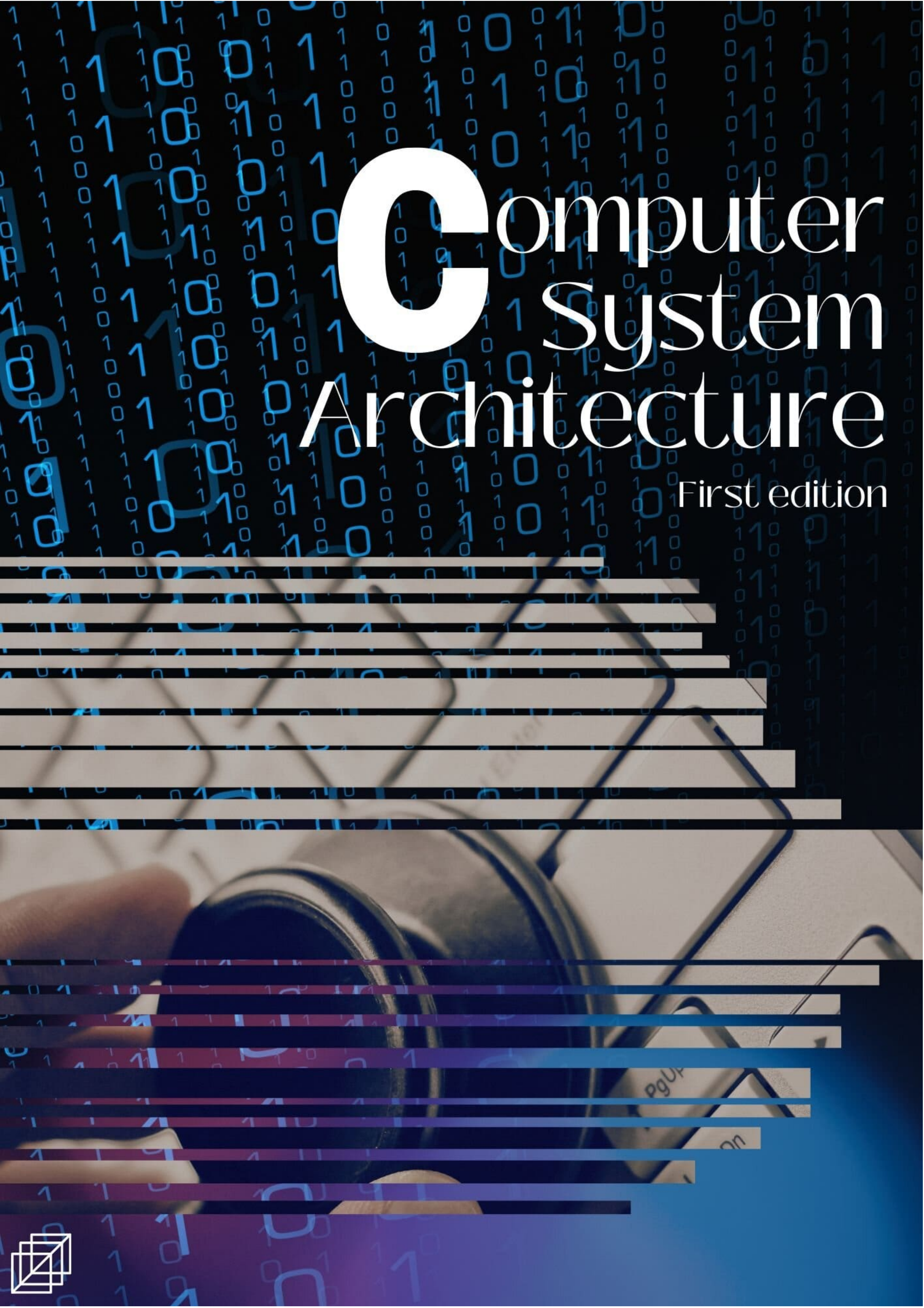
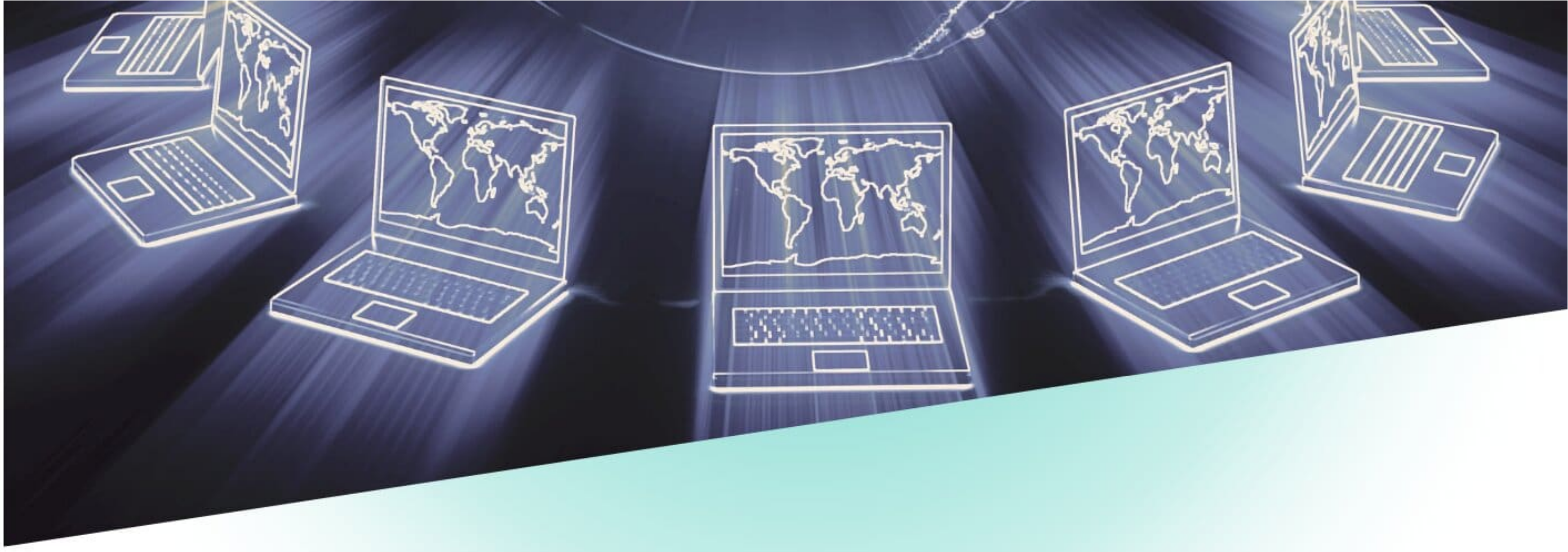


Computer System Architecture

First edition



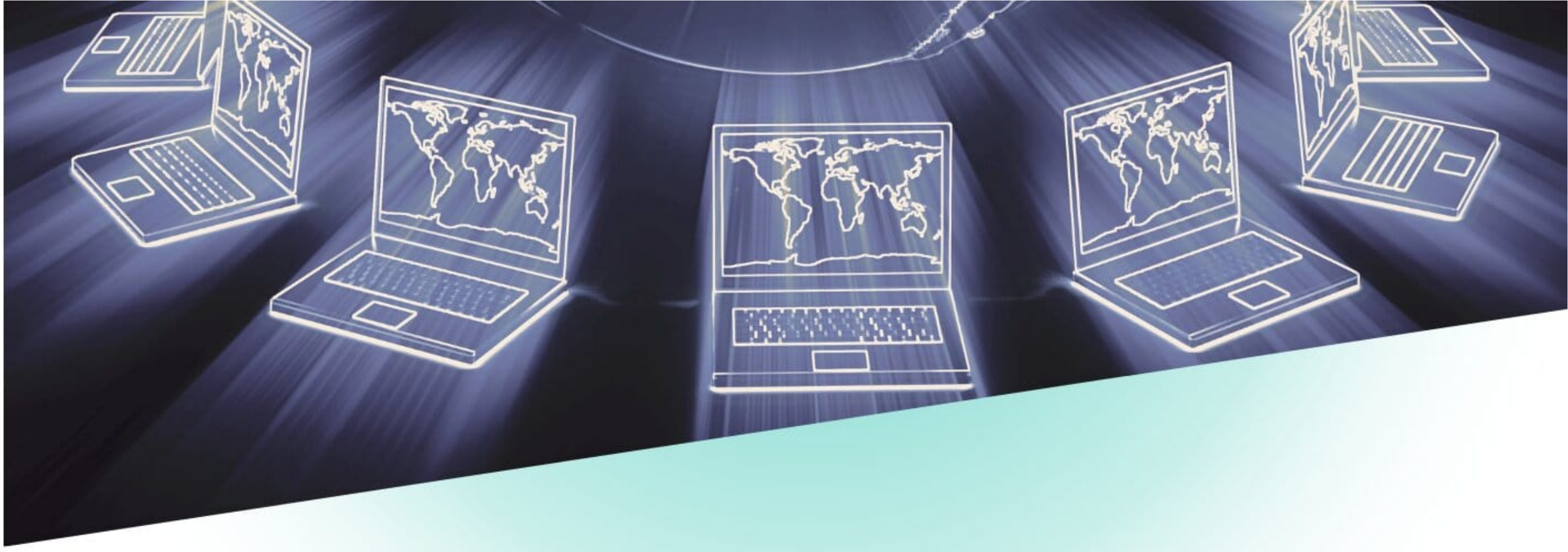
Computer System Architecture

1st Edition

Murniyati Binti Abdul

Department of Information Technology & Communication
Politeknik Sultan Mizan Zainal Abidin
Dungun, Terengganu.
2022

Pn



Perpustakaan Negara Malaysia Cataloguing-in-Publication Data

Murniyati Abdul, 1981-
Computer System Architecture / Murniyati Binti Abdul. – 1st Edition.
Mode of access: Internet
eISBN 978-967-0047-14-0
1. Computer systems.
2. Computer architecture.
3. Government publications--Malaysia.
4. Electronic books.
I. Title.
004.22

Pn

Preface

COMPUTER SYSTEM ARCHITECTURE course is designed to introduce the basic concepts on which the stored program digital computer is formulated. These include the introduction of computer architecture and computer organisation, and the representation and manipulation of numbering system. This goal addresses the question on how does a computer work and how it is organized. The course also provides students with foundation knowledge of the Central Processing Unit and assembly language programming.

Table of Contents

Pn

THE COMPUTER SYSTEM

THE BASIC CONCEPTS OF COMPUTER ARCHITECTURE

Introduction

- **Computer organization** is concerned with the way the hardware components operate and the way they are connected together to form the computer system. The various components are assumed to be in the place and the task is to investigate the organizational structure to verify that the computer parts operate as intended.

- **Computer design** is concerned with the hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system. Computer design is concerned with the determination of what hardware should be used and how the parts should be connected. This aspect of computer hardware is sometimes referred to as computer implementation.

- **Computer architecture** is concerned with the structure and behavior of the computer as seen by the user. It includes the information format, the instruction set and techniques for addressing memory. The architectural design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories and structuring them together into a computer system.

  (M. Morris Mano, 1993)

- A working computer requires hardware and software.
  - Hardware: the computer's physical and electronic components (input devices, a central processing unit (CPU), memory-storage devices and output devices). All components are linked together by a communication network or bus.
  - Software: the programs that instruct the hardware to perform tasks (operating system software and applications software).

Interconnection structures within computer system

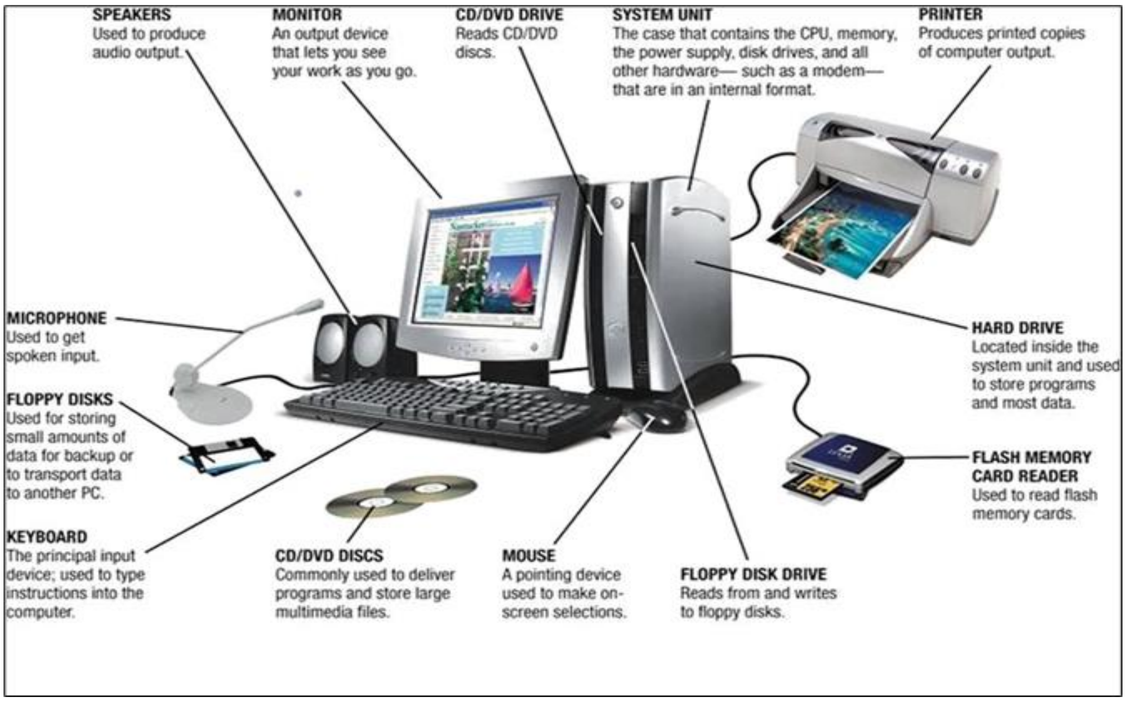

SPEAKERS
Used to produce audio output.
MONITOR
An output device that lets you see your work as you go.
CD/DVD DRIVE
Reads CD/DVD discs.
SYSTEM UNIT
The case that contains the CPU, memory, the power supply, disk drives, and all other hardware— such as a modem— that are in an internal format.
PRINTER
Produces printed copies of computer output.
MICROPHONE
Used to get spoken input.
FLOPPY DISKS
Used for storing small amounts of data for backup or to transport data to another PC.
HARD DRIVE
Located inside the system unit and used to store programs and most data.
FLASH MEMORY CARD READER
Used to read flash memory cards.
KEYBOARD
The principal input device; used to type instructions into the computer.
CD/DVD DISCS
Commonly used to deliver programs and store large multimedia files.
MOUSE
A pointing device used to make on-screen selections.
FLOPPY DISK DRIVE
Reads from and writes to floppy disks.

Figure 1.1: Components in micro computer system.

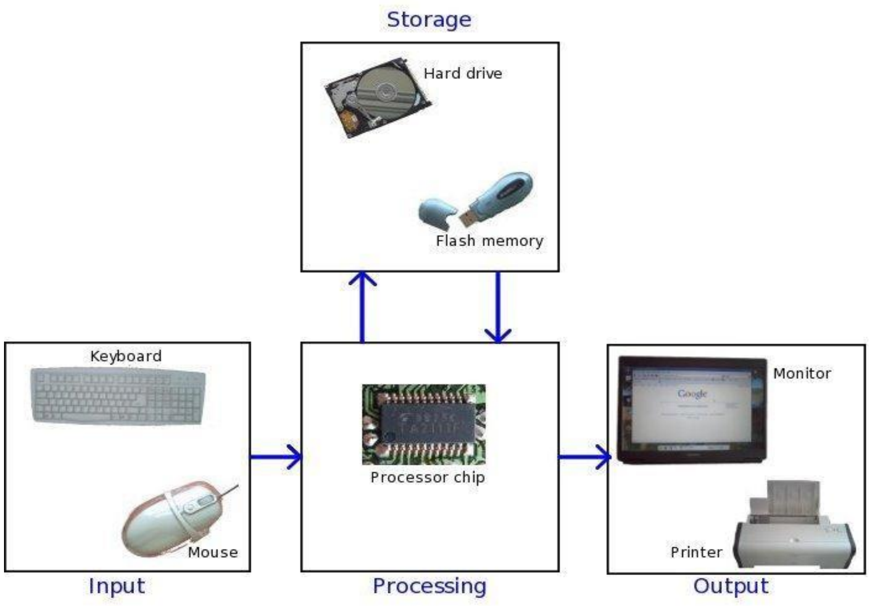


Figure 1.2: Interconnection structures in computer system.

1. Input - It accepts data or instructions by way of input
2. Storage - It stores data
3. Processing - It can process data as required by the user

4. Output - It gives results in the form of output
5. Control - It controls all operations inside a computer.

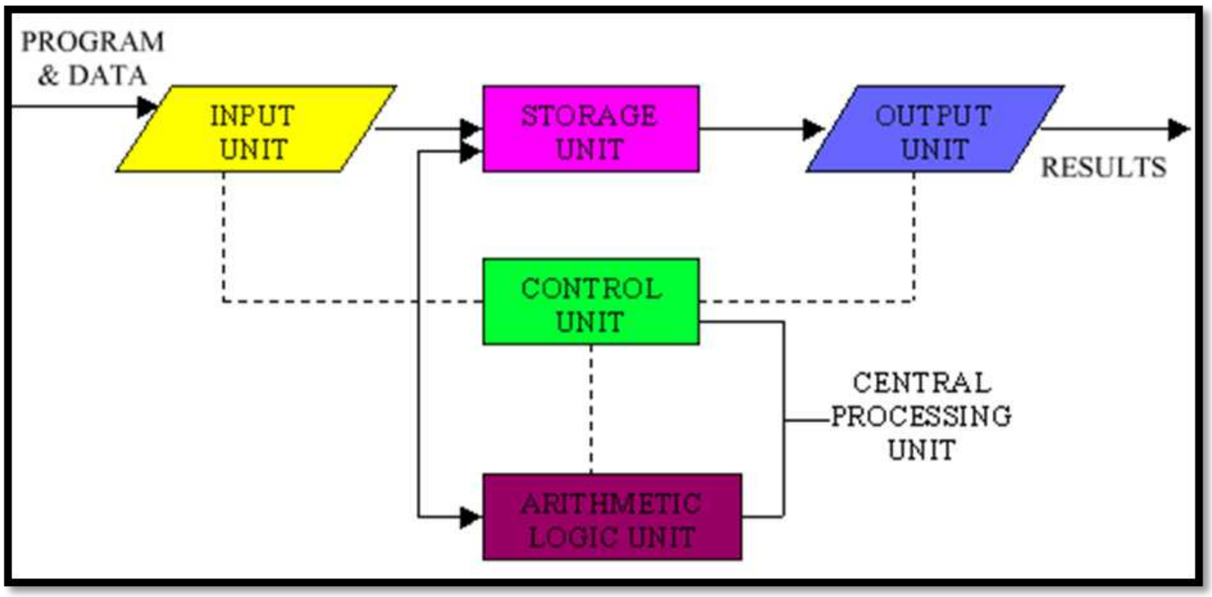


Figure 1.3: Basic computer operation

Input

- This is the process of entering data and programs in to the computer system.
- You should know that computer is an electronic machine like any other machine which takes as inputs raw data and performs some processing giving out processed data.
- Therefore, the input unit takes data to the computer in an organized manner for processing.

Storage

- The process of saving data and instructions permanently is known as storage.
- Data has to be fed into the system before the actual processing starts. It is because the processing speed of Central Processing Unit (CPU) is so fast that the data has to be provided to CPU with the same speed. Therefore the data is first stored in the storage unit for faster access and processing.
- This storage unit or the primary storage of the computer system is designed to do the above functionality. It provides space for storing data and instructions.
- The storage unit performs the following major functions:

- All data and instructions are stored here before and after processing.
- Intermediate results of processing are also stored here.

Processing

- The task of performing operations like arithmetic and logical operations is called processing.
- The Central Processing Unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided. It is then sent back to the storage unit.

Output

- This is the process of producing results from the data for getting useful information.
- Similarly the output produced by the computer after processing must also be kept somewhere inside the computer before being given to you in human readable form.
- Again the output is also stored inside the computer for further processing.

Control (operations inside the computer)

- The manner how instructions are executed and the above operations are performed.
- Controlling of all operations like input, processing and output are performed by control unit.
- It takes care of step by step processing of all operations inside the computer.

Three basic computer functional units control the operations of a computer

In order to carry out the operations mentioned in the previous section the computer allocates the task between its various functional units. The computer system is divided into three separate units for its operation.

They are:

a) Arithmetic and Logic Unit (ALU)
b) Control unit (CU)
c) Central processing unit (CPU)

Arithmetic and Logic Unit (ALU)

After you enter data through the input device it is stored in the primary storage unit. The actual processing of the data and instruction are performed by Arithmetic Logical Unit. The major operations performed by the ALU are addition, subtraction, multiplication, division, logic and comparison. Data is transferred to ALU from storage unit when required. After processing the output is returned back to storage unit for further processing or getting stored.

Control unit (CU)

The next component of computer is the Control Unit, which acts like the supervisor seeing that things are done in proper fashion. The control unit determines the sequence in which computer programs and instructions are executed. Things like processing of programs stored in the main memory, interpretation of the instructions and issuing of signals for other units of the computer to execute them. It also acts as a switch board operator when several users access the computer simultaneously. Thereby it coordinates the activities of computer's peripheral equipment as they perform the input and output. Therefore it is the manager of all operations mentioned in the previous section.

Central processing unit (CPU)

The ALU and the CU of a computer system are jointly known as the central processing unit. You may call CPU as the brain of any computer system. It is just like brain that takes all major decisions, makes all sorts of calculations and directs different parts of the computer functions by activating and controlling the operations.

Bus Interconnection

Block diagram to illustrate the basic organization of computer system

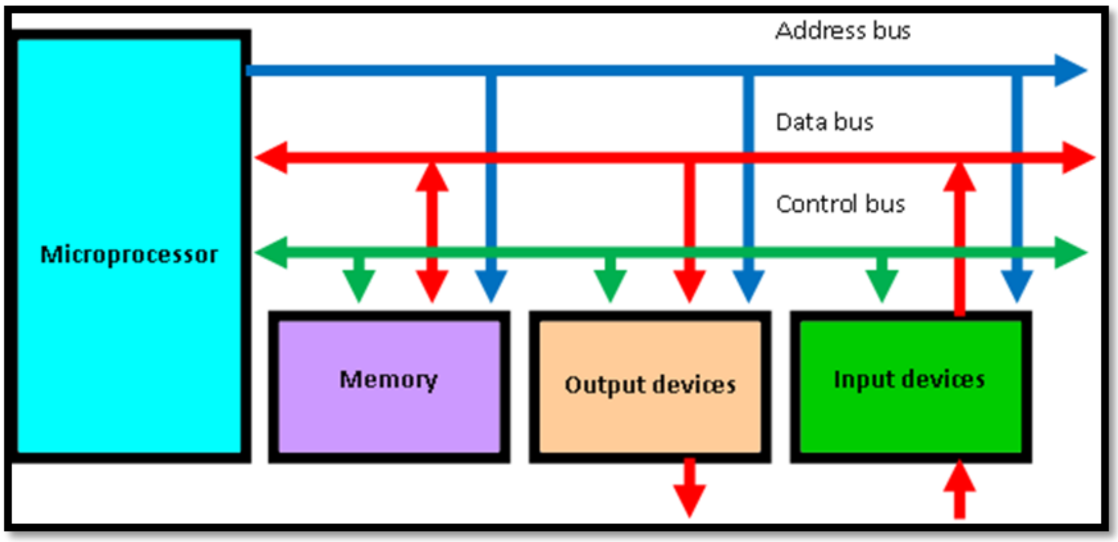


Figure 1.4: Block diagram for basic computer system.

A computer system consist four standard components; processor, memory, input / output and timing. An additional circuits may also be used depends on the application. This includes analog / digital converters, digital / analog converter, disk drive controller, video display, and etc.

All components are connected via a bus system that is shared by a group of conductive component in the system. Bus system is divided into address bus, data bus and control bus.

Processor

- Also known as Control Processing Unit (CPU).
- It controls the whole operation by a system.
- Processor will execute the instruction in computer memory.

Memory

- Is used to save programs and data required by processor.
- Memory can be divided into two types; RAM and ROM.
- The content of a ROM is permanent while the contents of a RAM will be lost when the power supply is disconnected.

Input/output

- Allow computer to communicate with the external environment.
- Serial I/O device enables a communication with the larger computer or terminal operator.
- Parallel I/O device involve switch reading operation and data transmission with other parallel device such as digital-analog converter, analog-digital converter or disk drive.

The Von Neumann model

- Von Neumann proposed this model in 1946.
- Program instructions and Data are *both* stored as sequences of bits in computer memory.
- Components of the Von Neumann Model
  1. **Memory**: Storage of information (data/program)
  2. **Processing Unit**: Computation/Processing of Information (performs the data processing)
  3. **Input**: Means of getting information into the computer. e.g. keyboard, mouse (to enter data and program)
  4. **Output**: Means of getting information out of the computer. e.g. printer, monitor (to extract results)
  5. **Control Unit**: Makes sure that all the other parts perform their tasks correctly and at the correct time.

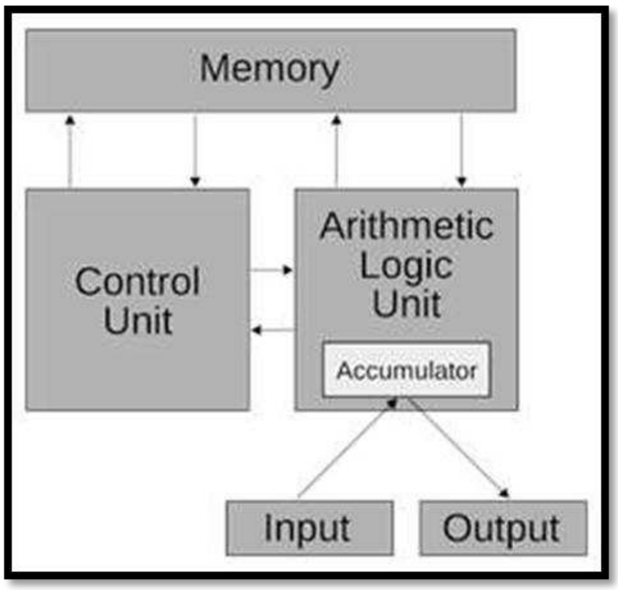


Figure 1.5: The Von Neumann model.

COMPUTER'S BUS SYSTEM.

Definition of computer's bus

A bus, in computing, is a set of physical connections (cables, printed circuits, etc.) which can be shared by multiple hardware components in order to communicate with one another.

A bus is characterized by the amount of information that can be transmitted at once. This amount, expressed in bits, corresponds to the number of physical lines over which data is sent simultaneously. In reality, each bus is divided into three subassemblies (refer Figure 1.3):

- The **address bus** (sometimes called the *memory bus*) transports memory addresses which the processor wants to access in order to read or write data. It is a unidirectional bus.
- The **data bus** transfers instructions coming from or going to the processor. It is a bidirectional bus.
- The **control bus** (or *command bus*) transports orders and synchronization signals coming from the control unit and travelling to all other hardware components. It is a bidirectional bus, as it also transmits response signals from the hardware.

Two types of computer's bus

There are generally two buses within a computer:

a) Internal bus (system bus)
b) External bus (expansion bus)

- **The internal bus** (sometimes called the *front-side bus*, *FSB*). An internal bus connects all the internal components of a computer to the motherboard (and thus, the CPU and internal memory). These types of buses are also referred to as a local bus
- **The external bus / expansion bus** (sometimes called the *input/output bus*) allows various motherboard components (USB, serial, and parallel ports, cards inserted in PCI connectors, hard drives, CD-ROM and CD-RW drives, etc.) to communicate with one

another. However, it is mainly used to add new devices using what are called **expansion slots** connected to the input/output bus.

Concept of Cache Memory

Introduction to cache memory

– Cache memory is a special high-speed storage mechanism. It can be either a reserved section of main memory or an independent high-speed storage device.

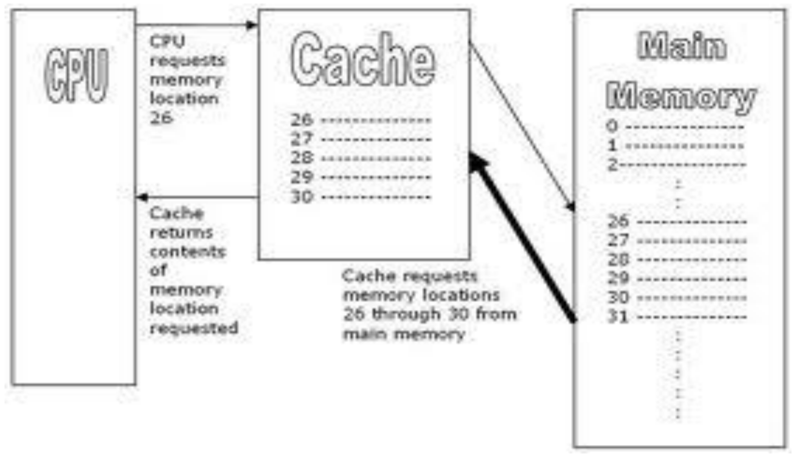

CPU
CPU requests memory location 26
Cache
26
27
28
29
30
Cache returns contents of memory location requested
Cache requests memory locations 26 through 30 from main memory
Main Memory
0
1
2
26
27
28
29
30
31

Figure 1.6: Cache and Main Memory

Types Of Cache Memory

The basic characteristic of cache memory is its fast access time. Therefore, very little or no time must be wasted when searching for words in the cache. The transformation of data from main memory to cache memory is refers to as a mapping process. Three types of mapping procedures are of practical interest when considering the organization of cache memory is direct, associative and set-associative mapped cache.

Direct Mapping

- Simplest technique
- maps each block of the main memory into one possible cache line

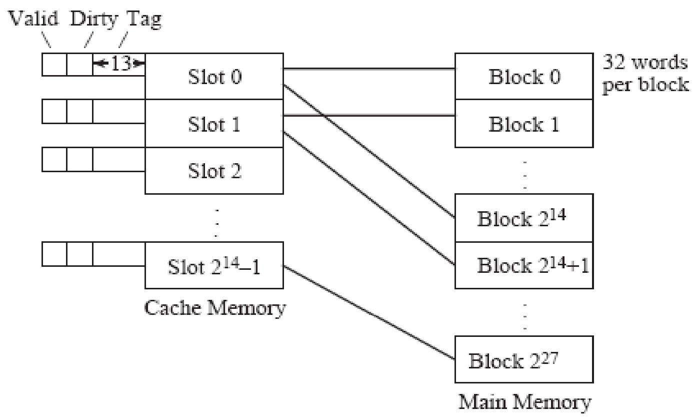


Figure 1.7: Direct Mapping

Associative Mapping

- Overcome the disadvantages of the direct mapped by permitting each main memory block to be loaded into any line of cache

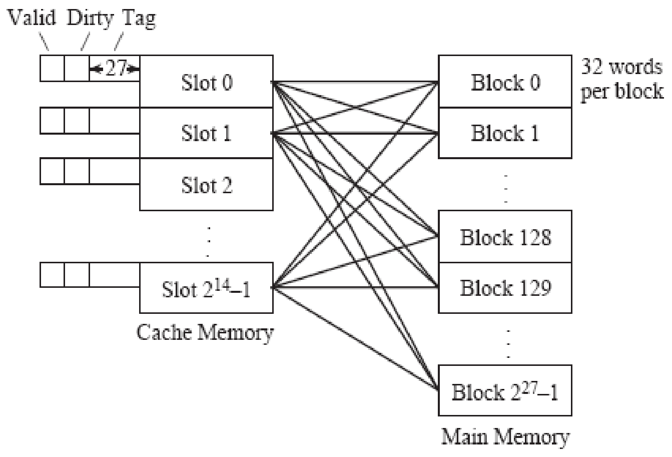


Figure 1.8: Associative Mapping

Set-associative Mapping

- Combines the strengths of both direct and associative approaching

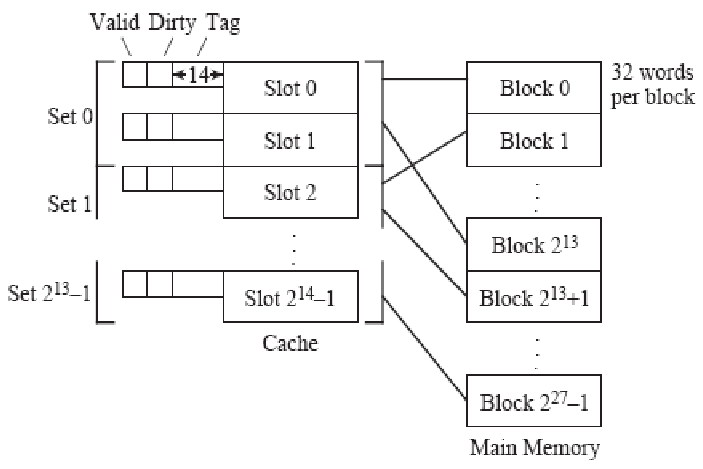


Figure 1.9: Set Associative Mapping

<u>Input/Output in Computer System</u>

- an element of computer system
- composed of two parts
    - Input unit-providing input to processor
    - Output unit-receiving output from processor
- Provide interaction with outside world

Block Diagram Input Output Unit

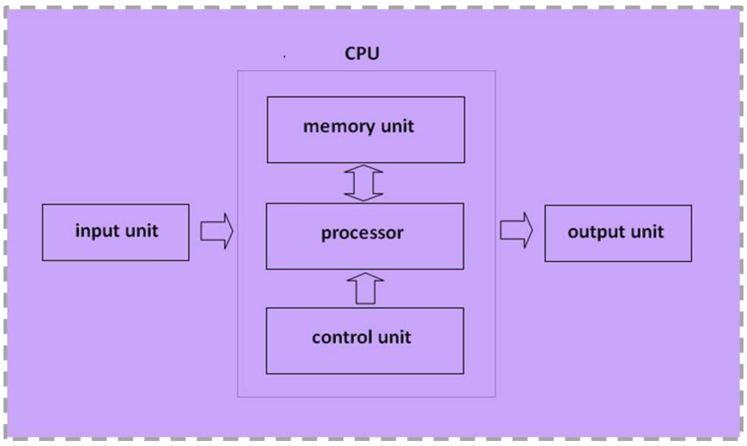


Figure 1.10: Block diagram Input Output Unit

Input Output Devices

- Input devices
  -human data entry devices and source entry devices
  -human entry devices(keyboard, mouse, joystick, digitizing tablet,..)
  -source entry devices(microphone, soundcard, video camera,…)

- Output devices
  -hardcopy and softcopy devices
  -hardcopy devices(printer, plotter)
  -softcopy devices(monitor, visual display terminal, video output,…)

Input Output Module Diagram

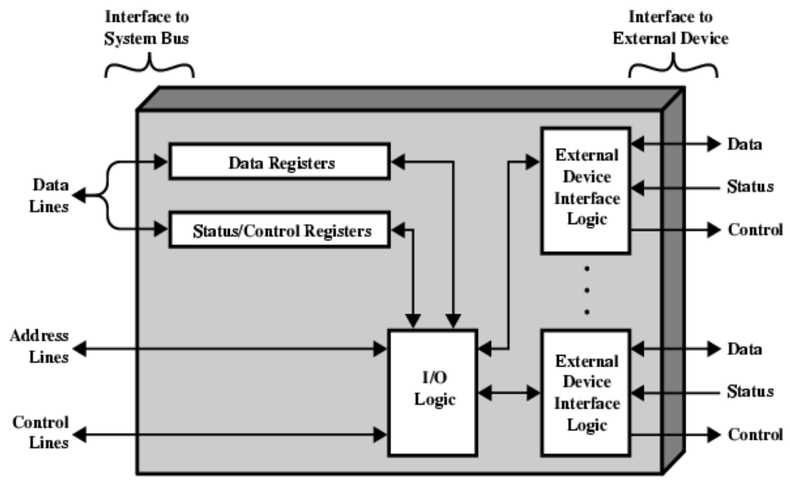
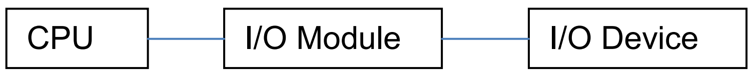


Figure 1.11: Input Output Module Diagram

Configuration through I/O Module



Generic Model of Input Output Module

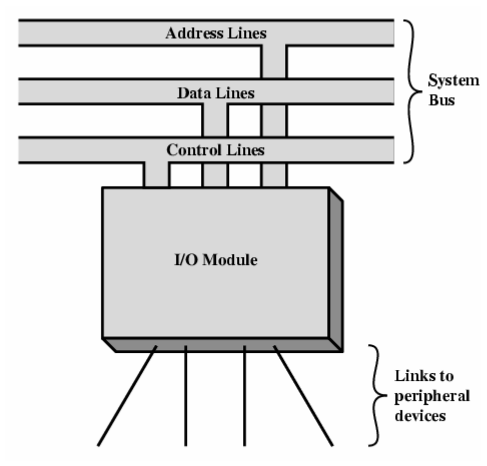


Figure 1.12: Model Input Output Module

Input Output Steps

1) CPU checks I/O module device status
2) I/O module returns status
3) If ready, CPU requests data transfer
4) I/O module gets data from device
5) I/O module transfers data to CPU

Input Output Data Transfer

Serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus.

- Data is transmitted one byte at one time
- Each frame contains 1 start bit,8 data bits, parity bit and 1 stop bit
- High amount of overhead(effect on the computer's performance)
- Utilizes a transmitter, a receiver and a wire without coordination about its clock to match the incoming signal

Input Output Techniques

Programmed I/O

- CPU has direct control over I/O
  - –Sensing status
  - –Read/write commands
  - –Transferring data
- CPU waits for I/O module to complete operation
- Waste the CPU time

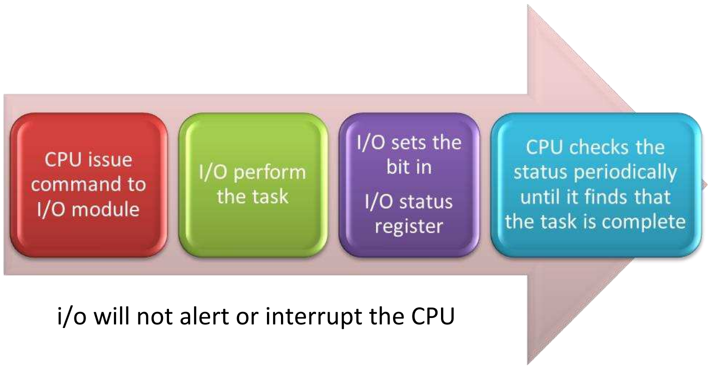

CPU issue command to I/O module
I/O perform the task
I/O sets the bit in I/O status register
CPU checks the status periodically until it finds that the task is complete
i/o will not alert or interrupt the CPU

Figure 1.13: Block diagram Programmed I/O

Interrupt-initiated I/O

- Able to overcome programmed I/O disadvantages
  - CPU have to wait for a long time for I/O module
- CPU issue an I/O command to a module and will go on to do other useful work. I/O will interrupt the CPU when it is ready for exchange data. The CPU execute the data transfer and resume to its former processing.

Direct Memory Access (DMA)

- Draw back of programmed and interrupt driven I/O
  - I/O transfer rate is limited
  - Processor is tied up in managing an I/O data transfer.
- When dealing with a large volume of data, more efficient method are required-DMA
- Transfer large amounts of data at high speed without continuous intervention by the processor
- Special control circuit required in the I/O device interface, called a DMA controller
- DMA controller keeps track of memory locations, transfers directly to memory (via the bus) independent of the processor
- when processor wishes to read/write, it send the command to DMA module by issuing information as:-
  - Direction of transfer,read(I/O memory) or write (memory I/O)

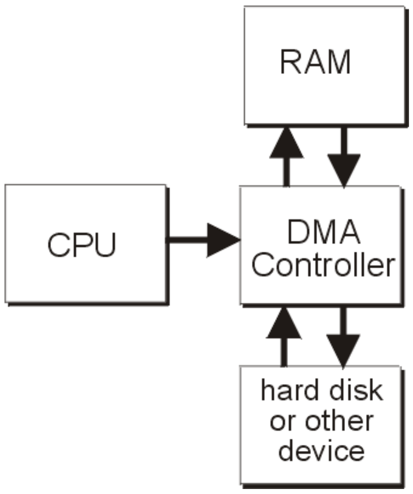
- Address of the I/O device involved
- The starting location of the block of the data in memory
- The size of the block to be transferred

➢ The processor continue with other work. It has delegated this DMA module.

➢ DMA module transfers the entire block of data, directly from memory without involving processor.

➢ After completing the data transfer, DMA modules sends interrupt signal to the processor.

➢ Processor only involved in beginning and end of the transfer



<u>How information is transferred over synchronous and asynchronous buses?</u>

Asynchronous and synchronous communication refers to methods by which signals are transferred in computing technology. These signals allow computers to transfer data between components within the computer or between the computer and an external network. Most actions and operations that take place in computers are carefully controlled and occur at specific times and intervals. Actions that are measured against a time reference, or a clock signal, are referred to as synchronous actions. Actions that are prompted as a response to another signal, typically not governed by a clock signal, are referred to as asynchronous signals.

Typical examples of synchronous signals include the transfer and retrieval of address information within a computer via the use of an address bus. For example, when a processor places an address on the address bus, it will hold it there for a specific period of time. Within this interval, a particular device inside the computer will identify itself as the one being addressed and acknowledge the commencement of an operation related to that address.

- **Synchronous Bus:**
  - Includes a clock in the control lines
  - A fixed protocol for communication: relative to the clock
  - Advantage: involves very little logic and can run very fast
  - Disadvantages:
    - Every device on the bus must run at the same clock rate
    - Clock skew => they cannot be long if they are fast

- **Asynchronous Bus:**
  - It is not clocked
  - It can accommodate a wide range of devices
  - It can be lengthened without worrying about clock skew
  - It requires a handshaking protocol

Common Bus Protocol; PCI And SCSI.

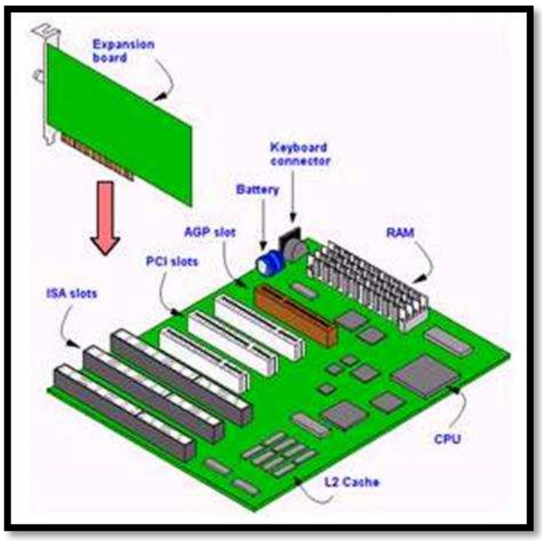


Figure 1.14: Location of PCI and SCSI slot on a motherboard.

Peripheral Component Interconnect (PCI)

PCI is a computer bus for attaching hardware devices in a computer. These devices can take either the form of an integrated circuit fitted onto the motherboard itself, called a *planar device* in the PCI specification, or an expansion card that fits into a slot.

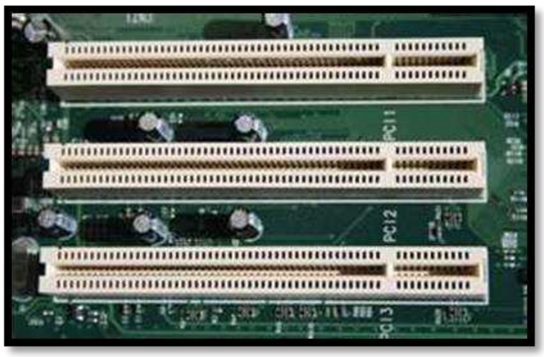


Figure 1.15: PCI Slot

Many devices traditionally provided on expansion cards are now commonly integrated onto the motherboard itself, meaning that modern PCs often have no cards fitted. However, PCI is still used for certain specialized cards; although many tasks traditionally performed by expansion cards may now be performed equally well by USB devices.

Small Computer System Interface (SCSI)

SCSI is a set of standards for physically connecting and transferring data between computers and peripheral devices. The SCSI standards define commands, protocols, and electrical and optical interfaces. SCSI is most commonly used for hard disks and tape drives, but it can connect a wide range of other devices, including scanners and CD drives.

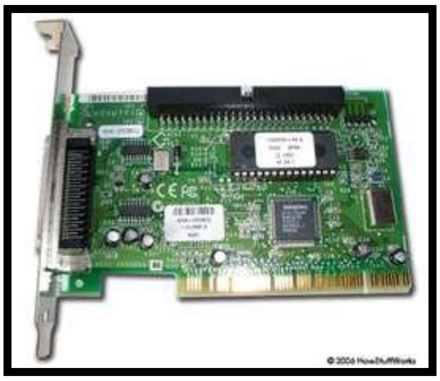
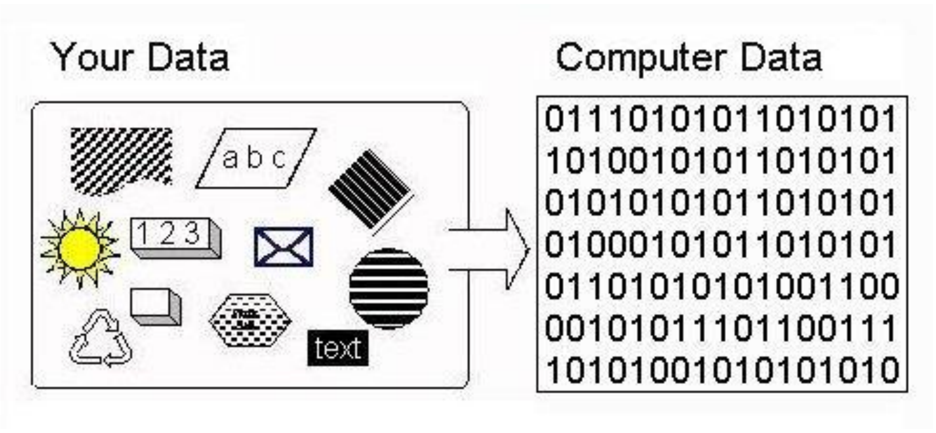


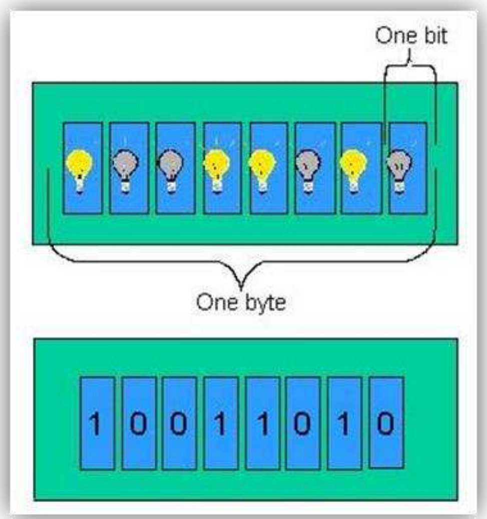
Figure 1.16: Adapter SCSI Card

ARITHMETIC AND LOGIC

DATA REPRESENTATION ON CPU

Humans are accustomed to dealing with decimal numbers, while computers use binary digits. Octal and hexadecimal numbers are "short forms" for binary numbers, where each hexadecimal digit takes the place of either three or four binary digits. Since people and computers speak different "number languages", it is often necessary to convert numbers from one of these systems to the other. If you spend any amount of time dealing with computers or networks, you will find yourself needing to do this on occasion, so it's worth taking a quick look at how it is done.





Decimal, Binary, Octal and Hexadecimal Number

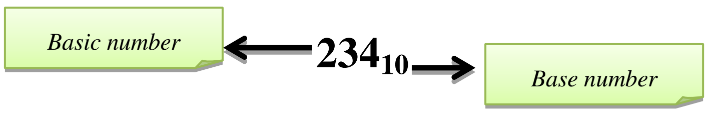
There are 4 types of numbering system used in digital system.

a. Decimal system (N_{10})

b. Octal system (N_8)

c. Hexadecimal system (N_{16})

d. Binary system (N_2)



a. Decimal system (N_{10})

- ✖ Base 10
- ✖ Use ten digits
- ✖ Number 0,1,2,3,4,5,6,7,8,9

Table 1 : Decimal weights

	10^2	10^1	10^0	.	10^{-1}	10^{-2}
weights	100	10	1	.	0.1	0.01

b. Octal system (N_8)

- ✖ Base 8
- ✖ Use eight digits
- ✖ Number 0,1,2,3,4,5,6,7

Table 2 : Octal weights

	8^2	8^1	8^0	.	8^{-1}	8^{-2}
weights	64	8	1	.	0.125	0.015625

c. Hexadecimal system (N_{16})

- Base 16
- Use sixteen digits
- Number 0,1,2,3,4,5,6,7,8,9
  A,B,C,D,E,F (A=10, B=11, C=12, D=13, E=14, F=15)

Table 3 : Hexadecimal weights

	16^2	16^1	16^0	.	16^{-1}	16^{-2}
weights	256	16	1	.	0.0625	0.0039

d. Binary system (N_2)

- Base 2
- Use two digits
- 0 and 1

Table 4 : Binary weights

	2^2	2^1	2^0	.	2^{-1}	2^{-2}
weights	4	2	1	.	0.5	0.25

Table 5: Computer Numbering System

Binary Digits	**Octal Digit**	**Hexadecimal Digit**	**Decimal Digit**
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6

0111	7	7	7
1000		8	8
1001		9	9
1010		A	
1011		B	
1100		C	
1101		D	
1110		E	
1111		F	

Arithmetic Operation in Different Number Bases

Addition and subtraction rules for binary number

Basic rules of **addition**	Basic rules of **subtraction**
0 + 0 = 0	0 - 0 = 0
0 + 1 = 1	0 - 1 = 1 with a borrow of 1
1 + 0 = 1	1 - 0 = 1
1 + 1 = 0 with a carry of 1	1 - 1 = 0

a) Addition

Decimal number

Example:

Add the following decimal number:

i. $25_{10} + 87_{10}$

$$\begin{array}{rrr} & {}^{1} & {}^{1} \\ & 2 & 5_{10} \\ + & 8 & 7_{10} \\ \hline & \mathbf{1\,1} & \mathbf{2_{10}} \\ \hline\hline \end{array}$$

step1: $5 + 7 = \underbrace{12-10} = 2$

step2: $1 + 2 + 8 = \underbrace{11-10} = 1$

step3

Exercise :

i) $195_{10} + 23_{10}$

ii) $892_{10} + 158_{10}$

Octal number

Example:

Add the following octal number:

i. $356_8 + 124_8$

$$\begin{array}{rrrr} & {}^{1} & {}^{1} & \\ & 3 & 5 & 6_8 \\ + & 1 & 2 & 4_8 \\ \hline & \mathbf{5} & \mathbf{0} & \mathbf{2_8} \\ \hline\hline \end{array}$$

step1: $6 + 4 = \underbrace{10-8} = 2$

step2: $1 + 5 + 2 = \underbrace{8-8} = 0$

step3: $1 + 3 + 1 = 5$

Exercise :

i) $123_8 + 321_8$

ii) $733_8 + 74_8$

Hexadecimal number

Example:

Add the following hexadecimal number:

i. $C1_{16} + E_{16}$

$$\begin{array}{rcc} & C & 1_{16} \\ + & & E_{16} \\ \hline & \mathbf{C} & \mathbf{F_{16}} \\ \hline\hline \end{array}$$

step1: 1 + 14 (E) = 15 (F)

step2: C

Exercise :

i) $2B_{16} + 84_{16}$

ii) $DF_{16} + AC_{16}$

Binary number

Example:

Add the following binary number:

i. $101_2 + 11_2$

$$\begin{array}{rcccc} & {}^{1} & {}^{1} & & {}^{1} \\ & & 1 & 0 & 1_2 \\ + & & & 1 & 1_2 \\ \hline & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0_2} \\ \hline\hline \end{array}$$

step1: 1 + 1 = 0 carry 1

step2: 1 + 0 + 1 = 0 carry 1

step3: 1 + 1 =0 carry 1

step4: 1

Exercise :

i. $1010_2 + 0111_2$

ii. $10111_2 + 111_2$

b) Subtraction

Decimal number

Example:

Sub the following decimal number:

i. $73_{10} - 19_{10} = \mathbf{54_{10}}$.

$$\begin{array}{rrrl} & \overset{6}{\cancel{7}} & \overset{10}{3} & _{10} \\ - & 1 & 9 & _{10} \\ \hline & \mathbf{5} & \mathbf{4} & _{\mathbf{10}} \\ \hline\hline \end{array}$$

step1: 10 + 3 = 13 – 9 = 4

step2: 6 – 1 = 5

Exercise :

i. $254_{10} - 46_{10}$

ii. $192_{10} - 15_{10}$

Octal number

Example:

Sub the following octal number:

i. $62_8 - 53_8 = \mathbf{07_8}$

$$\begin{array}{rrrl} & \overset{5}{\cancel{6}} & \overset{8}{2} & _{8} \\ - & 5 & 3 & _{8} \\ \hline & 0 & 7 & _{8} \\ \hline\hline \end{array}$$

step1: 8 + 2 =10-3 = 7

step2: 0

Exercise :

i. $1523_8 - 364_8$

ii. $145_8 - 64_8$

Hexadecimal number

Example:

Sub the following octal number:

i. $B2_{16} - B_{16} = \mathbf{A7_{16}}$

$$\begin{array}{rcc} & \overset{A}{\cancel{B}} & \overset{16}{2_{16}} \\ - & & B_{16} \\ \hline & A & 7_{16} \\ \hline \hline \end{array}$$

step1: 16+ 2 = 18-B(11) =7

step2: A

Exercise :

i. $84_{16} - 2A_{16}$

ii. $A2D_{16} - 5F_{16}$

Binary number

Example:

Sub the following binary number:

i. $101_2 - 011_2 = 010_2$

$$\begin{array}{rccc} & \overset{0}{\cancel{1}} & 0 & 1_2 \\ - & 0 & 1 & 1_2 \\ \hline & 0 & 1 & 0_2 \\ \hline \hline \end{array}$$

step1: 1 - 1 = 0

step2 : 0-1 = 1(borrow 1)

step3 : 0 – 0 = 0

Exercise :

i. $100_2 - 1_2$

ii. $101011_2 - 1111_2$

Numbering system conversion

Binary-To-Decimal Conversion

- To express the value of a given binary number as its decimal equivalent, we just need to sum the digits after each has been multiplied by its associated weight

Exercise :

i) 110101_2

ii) 0.1011_2

Example:

Convert 100.010_2 to decimal number.

Solution:

Weight	: 2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3}
Binary number	: 1 0 0 . 0 1 0

$$100.010_2 = (1 \text{ X } 2^2) + (1 \text{ X } 2^{-2})$$
$$= 4 + 0.25$$
$$= 4.25_{10}$$

LSB
The Least Significant Bit is the rightmost digit which has lowest weight of a given number

MSB
The Most Significant Bit is the leftmost binary digit which has highest weight of a given number

Decimal-To- Binary Conversion

- To convert decimal to binary use this approach :
  - Divide the decimal value by two and record the remainder
  - Repeat first step until the decimal value is equal to zero
  - The first remainder produced is the LSB in the binary number and the last remainder is the MSB.

Example:

Convert 21.13_{10} to binary number.

Exercise :

i) 18_{10}

ii) 24.32_{10}

Solution:

2	21			
2	10	-	1	↑
2	5	-	0	↑
2	2	-	1	↑
2	1	-	0	↑
	0	-	1	

0.13	X	2	=	0.26	-	0	↓
0.26	X	2	=	0.52	-	0	↓
0.52	X	2	=	1.04	-	1	

$21.13_{10} = 10101.001_2$

Octal-To-Decimal Conversion

Example:

Convert 372.24_8 to decimal number.

Exercise :

i) 25_8

ii) 237.04_8

Solution:

Weight	: 8^2 8^1 8^0 8^{-1} 8^{-2}
Octal number	: 3 7 2 2 4

$$372.24_8 = (3 \text{ X } 8^2) + (7 \text{ X } 8^1) + (2 \text{ X } 8^0) + (2 \text{ X } 8^{-1}) + (4 \text{ X } 8^{-2})$$

$$= 192 + 56 + 2 + 0.25 + 0.0625$$

$$= 250.3125_{10}$$

Decimal-To- Octal Conversion

Example:

Convert 82.7_{10} to octal number.

Exercise :

i) 58_{10}

ii) 300.34_{10}

Solution:

8	82			
8	10	-	2	↑
8	1	-	2	
	0	-	1	

0.7	X	8	=	5.6	-	5	↓
0.6	X	8	=	4.8	-	4	
0.8	X	8	=	6.4	-	6	

$82.7_{10} = 122.546_8$

Hexadecimal-To-Decimal Conversion

Exercise :

i) $AF2_{16}$

ii) $25E8_{16}$

Example:

Convert $7E7.7_{16}$ to decimal number.

Solution:

Weight	: 16^2 16^1 16^0 16^{-1}
Hexadecimal number	: 7 E 7 7

$$\begin{aligned} 7E7.7_{16} &= (7 \text{ X } 16^2) + (E \text{ X } 16^1) + (7\text{X } 16^0) + (7 \text{ X } 16^{-1}) \\ &= 1792 + 224 + 7 + 0.4375 \\ &= 2023.4375_{10} \end{aligned}$$

Decimal-To- Hexadecimal Conversion

Exercise :

i) 1375_{10}

ii) 650.20_{10}

Example:

Convert 2748.78_{10} to hexadecimal number.

Solution:

16	2748			
16	171	-	12(C)	↑
16	10	-	11 (B)	
	0	-	10(A)	

0.78	X	16	=	12.48	-	12(C)	↓
0.48	X	16	=	7.68	-	7	
0.68	X	16	=	10.88	-	10(A)	

$2748.78_{10} = ABC.C7A_{16}$

Binary-To-Octal Conversion

Exercise :

i) 01.01110_2

ii) 1101.10001110_2

Each octal digit is represented by a 3 bit binary digit.

Example:

Convert 1100011_2 to octal number.

Solution:

421	421	421
001	100	011
1	4	3

$1100011_2 = 143_8$

Octal-To- Binary Conversion

- One (1) octal digit can be represented by three digit binary number.

Example:

Convert 25_8 to binary number.

Exercise :

i) 12.5_8

ii) 37.12_8

Solution:

2	5
421	421
010	101

$25_8 = 010101_2$

Binary-To-Hexadecimal Conversion

- Break the binary digits into groups of four starting from LSB.
- It may be necessary to add a zero as the MSB in order to complete a grouping of four digits.

Example:

Convert 101_2 to hexadecimal number.

Exercise :

i) 1010111_2

ii) 11.100011110_2

Solution:

8 4 2 1

1 0 1

5

$101_2 = 5_{16}$

Hexadecimal-To- Binary Conversion

- One (1) hexadecimal digit can be represented by four digit binary number.

Example:

Convert $3A_{16}$ to binary number.

Exercise :

i) $FB17_{16}$

ii) $12D.2_{16}$

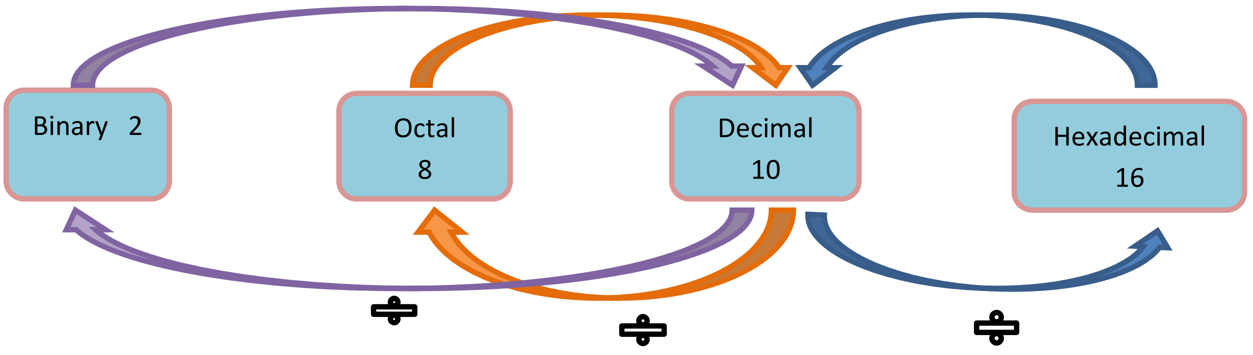
Solution:

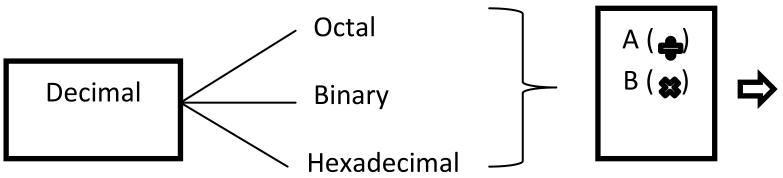
3 A

8 4 2 1 8 4 2 1

0011 1010

$3A_{16} = 111010_2$





EXAMPLE: $\underbrace{2748}_{A}.\underbrace{78}_{B}$

Binary to Octal

Octal to Binary

Binary to Hexadecimal

Hexadecimal to Binary

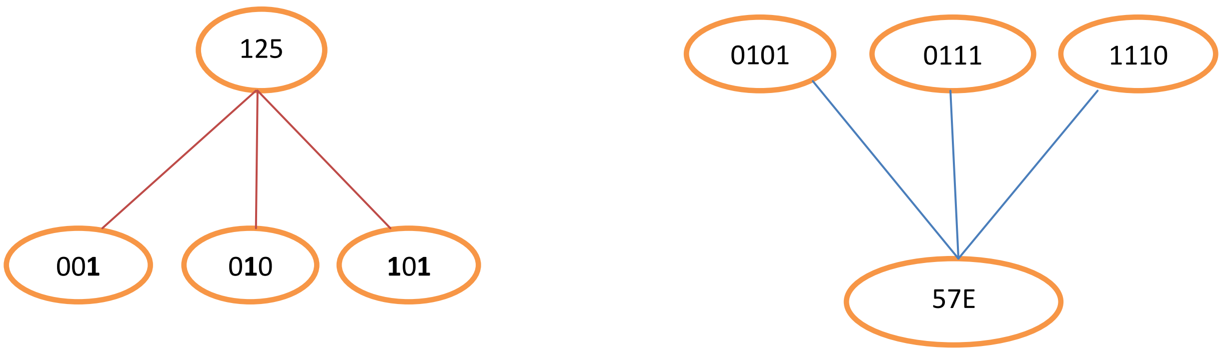
Use this Formula :

$\boxed{\textbf{8 4 2 18}}$

EXAMPLE:

1 father 3 son

3 son 1 father



1	2	5	5	7	14(E)
42**1**	4**2**1	**421**	8421	8**421**	**842**1

Coding System

a) Sign and magnitude

The left most bit is the sign bit and the remaining bit are the magnitude bits. The magnitude bit is in true binary for both positive and negative numbers.

Example:

Express the decimal number -39 as an 8 bit number in the sign-magnitude, 1st Complement and 2nd Complement Form.

Solution:

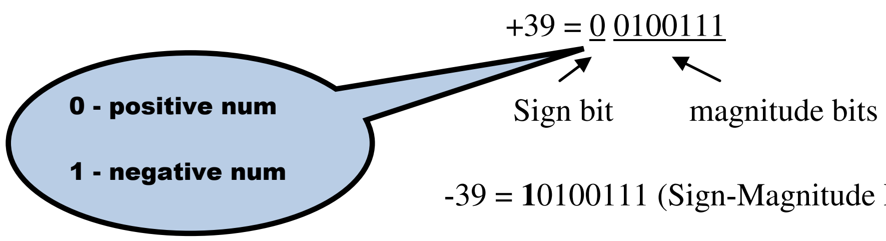
First, write the 8 bit number of +39

$+39 = 00100111_2$

Exercise :

i) -25_{10}
ii) $+70_{10}$

In the Sign-Magnitude Form, -39 is produced by changing the sign bits to a 1 and leaving the magnitude as they are.



-39 = **1**0100111 (Sign-Magnitude Form)

b) 1's Complement and 2's Complement

- Positive numbers in 1st Complement Form are represented the same way as in Signs Magnitude Form. Negative numbers, however, are the 1st Complement of the corresponding positive number.

- In the 1st Complement Form, -39 is produced by taking the 1st Complement of +39 (00100111) and changing bit 0 into 1 and bit 1 into 0.
- Use only in binary number

Exercise :

i. 32_{10}
ii. 114_{10}
iii. $11001_2 - 10011_2$

+39 = 00100111
= 11011000 (1st Complement Form)

Example:

Add the following number using 1st Complement Form

$8_{10} + (-3_{10})$

i. Convert 8_{10} and -3_{10} into N_2

$$
\begin{array}{lclcl}
8_{10} & \rightarrow & 1000_2 & & \\
-3_{10} & \rightarrow & +3 & \rightarrow & 0011_2 \\
 & \rightarrow & -3 & \rightarrow & 1100_2 \text{ (1st Complement)}
\end{array}
$$

ii. Solve $8_{10} + (-3_{10})$

$$
\begin{array}{cccccc}
 & & 1 & 0 & 0 & 0 \\
 & + & 1 & 1 & 0 & 0 \\
\hline
 & \textcircled{1} & 0 & 1 & 0 & 0 \\
+ & & \longrightarrow & & & 1 \\
\hline
 & & 0 & 1 & 0 & 1 \\
\hline
 & & {\scriptstyle 8} & {\scriptstyle 4} & {\scriptstyle 2} & {\scriptstyle 1}
\end{array}
\quad = \quad 5_{10}
$$

OR

$$
\begin{array}{lclcl}
8_{10} + (-3_{10}) & = & 8 & - & 3 \\
 & = & 5_{10} & & \\
 & & 0101_2 & &
\end{array}
$$

c) 2nd Complement Form

- Positive numbers in 2nd Complement Form are represented the same way as in Signs Magnitude Form and 1st Complement Form. Negative numbers are the 2nd Complement of the corresponding positive number.
- It is obtained by adding 1 to the LSB of the 1's complement value.
- In the 2nd Complement Form, -39 is produced by taking the 2nd Complement of +39 as follows.

$$\boxed{\text{2's complement} = \text{1's complement} + 1}$$

```
  11011000 (1st Complement Form)
+        1
  11011001 (2nd Complement Form)
```

Example:
Add the following number using 2nd Complement Form
$8_{10} + (-3_{10})$

Exercise :

i. 10011001_2
ii. 27_{10}

i. Convert 8_{10} and -3_{10} into N_2

$8_{10} \rightarrow 1000_2$
$-3_{10} \rightarrow +3 \rightarrow 0011_2$
$\rightarrow -3 \rightarrow 1100_2$ (1st Complement)

```
                 1  1  0  0
        +                 1
2nd
Complement       1  1  0  1₂
```

ii. Solve $8_{10} + (-3_{10})$

```
            1  0  0  0
         +  1  1  0  1
Ignore ← 1  0  1  0  1  =  5₁₀
            8  4  2  1
```

d) Binary Coded Decimal (BCD 8421 Code)

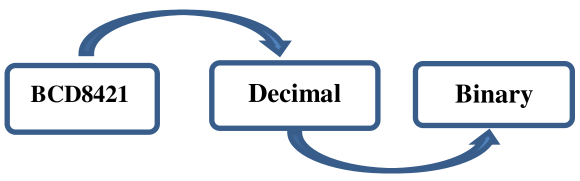
- A way to express each of the decimal digits with a binary code.
- Binary Coded Decimal means that each decimal digit, 0 through 9 is represented by a binary code of four bits.
- The designation 8421 indicates the binary weights of the four bits (2^3, 2^2, 2^1, 2^0).
- Invalid codes : 1010, 1011, 1100, 1101 , 1110 , 1111

Table 2: Decimal digit and BCD 8421 Code

Decimal Digit	BDC 8421
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
. .	. .
24	0010

BCD 8421-To-Binary Conversion

Example:
Convert 1001 0110$_{BCD8421}$ to binary number.



i. Convert BCD 8421 → N_{10}

$$\underbrace{1001}_{9} \quad \underbrace{0110}_{6} \rightarrow 96_{10}$$

ii. $96_{10} \rightarrow N_2$

2	96		
2	48	-	0
2	24	-	0
2	12	-	0
2	6	-	0
2	3	-	0
2	1	-	1
	0	-	1

$96_{10} = 1100000_2$

Exercise :

i. $1000\ 0011_{BCD8421}$

$$\begin{tikzcd}
\boxed{\textbf{Binary}} \arrow[r, bend left] & \boxed{\textbf{Decimal}} \arrow[r, bend right] & \boxed{\textbf{BCD8421}}
\end{tikzcd}$$

Binary-To-BCD 8421Conversion

Example:

Convert 1001010_2 to BCD 8421 Code.

Exercise :

i. $1100\ 1010_2$

i. Convert $N_2 \rightarrow N_{10}$

$$1001010_2 = (1 \times 2^6) + (1 \times 2^3) + (1 \times 2^1)$$
$$= 64 + 8 + 2$$
$$= 74_{10}$$

ii. Convert $N_{10} \rightarrow N_{BCD8421}$

$$74_{10} \rightarrow \underbrace{7}_{0111}\ \underbrace{4}_{0100} \rightarrow 01110100_{BCD8421}$$

e) ASCII Code

ASCII Code

– American Standard Code for Information interchange

- Is an alphanumeric code used in most computers and other electronic equipment.
- ASCII has 128 characters and symbols represented by a 7 bit binary code.
- Represent number, alphabet and symbol.

1

Table 3: ASCII Code

2

	$X_6X_5X_4$					
$X_3X_2X_1X_0$	010	011	100	101	110	111
0000	SP	0	@	P	`	p
0001	!	1	A	Q	a	q
0010	“	2	B	R	b	r
0011	#	3	C	S	c	s
0100	$	4	D	T	d	t
0101	%	5	E	U	e	u
0110	&	6	F	V	f	v
0111	‘	7	G	W	g	w
1000	(	8	H	X	h	x
1001	)	9	I	Y	i	y
1010	*	:	J	Z	j	z
1011	+	;	K	[	k	{
1100	,	<	L	\	l	|
1101	-	=	M	]	m	}
1110	.	>	N	^	n	~
1111	/	?	O	_	o	DEL

Example 1:

Convert GOTO 25 into ASCII Code

Solution:

G	-	1000111
O	-	1001111
T	-	1010100
O	-	1001111
SP		0100000
2	-	0110010
5	-	0110101

Exercise :

i. DIP2-S1

ii. 1010000 1001111
1001100 1001001

Example 2:

Message below are represented in ASCII code. What is the message.

1000100 1101001 1100111 1101001 1110100

BOOLEAN ALGEBRA

Logic Gates

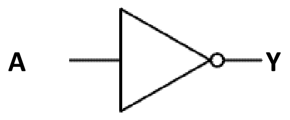
- A logic gate is an electronic circuit / device which makes the logical decisions.
- All other logic functions can ultimately be derived from combinations of these three.
- Digital (logic) circuit operate in the binary mode where each input and output voltage is either a 0 or a 1: the 0 and 1 designation represent predefined voltage ranges. In electronic digital, it's known as 'gate'.
- This characteristic of logic circuits allows us to use the Boolean Algebra as a tool for the analysis and design of digital systems.
- Boolean Algebra is a relatively simple mathematical tool that allows us to describe the relationship between logic circuits output and its input as an algebraic equation (a Boolean Expression).

- 7 types of logic gate are:
  i) NOT Gate (Inverter)
  ii) AND Gate
  iii) OR Gate
  iv) NAND Gate
  v) NOR Gate
  vi) Exclusive-OR Gate (Ex-OR)
  vii) Exclusive-NOR Gate (Ex-NOR)

a) Logic Gates Operation

a) NOT Gate (Inverter)

- The Inverter (NOT Gate) performs the operation called inversion or complementation.
- It changes one logic level to the opposite level. In terms of bit, it changes a 1 to a 0 and a 0 to a 1.
- Standard logic symbol for inverter



- Inverter truth table

○ Total number of possible input (2^n) $2^n = 2^1$ $= 2$	○ Maximum value ($2^n - 1$) $2^n - 1 = 2^1 - 1$ $= 2 - 1$ $= 1$

Input	**Output**
A	**Y**
0	1
1	0

- Boolean/Logic Expression

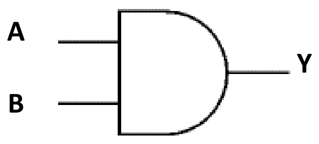
$$Y = \overline{A}$$

b) AND Gate

– An AND gate can have two or more inputs but only one output.

– Its output is true if all inputs are true.

– Standard logic symbol for AND Gate



- Truth table for a 2-input AND Gate

○ Total number of possible input (2^n) $2^2 = 2^2$ $= 4$	○ Maximum value ($2^n - 1$) $2^n - 1 = 2^2 - 1$ $= 4 - 1$ $= 3$

Input		Output
A	**B**	**Y**
0	0	0
0	1	0
1	0	0
1	1	1

- Boolean/Logic Expression

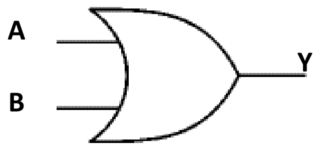
$$Y = A.B = AB$$

c) OR Gate

– An OR gate can have two or more inputs but only one output.

– Its output is true if at least one input is true.

– Standard logic symbol for OR Gate



- Truth table for a 2-input OR Gate

○ Total number of possible input (2^n) $2^2 = 2^2$ $= 4$	○ Maximum value ($2^n - 1$) $2^n - 1 = 2^2 - 1$ $= 4 - 1$ $= 3$

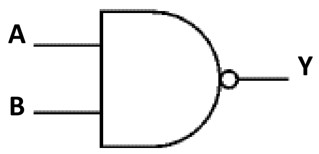
Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

- Boolean/Logic Expression

  $Y = A + B$

d) NAND Gate

– This is an AND gate with the output inverted, as shown by the 'o' on the output.
– A NAND gate can have two or more inputs.
– Its output is true if NOT all inputs are true.
– Standard logic symbol for NAND Gate



- Truth table for a 2-input NAND Gate
  - Total number of possible input (2^n) = 4
  - Maximum value (2^n – 1) = 3

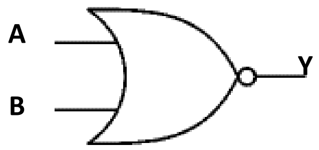
Input		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

- Boolean/Logic Expression

  $Y = \overline{A.B}$

e) NOR Gate

- This is an OR gate with the output inverted, as shown by the 'o' on the output.
- A NOR gate can have two or more inputs.
- Its output is true if no inputs are true.
- Standard logic symbol for NOR Gate



- Truth table for a 2-input NOR Gate
  - Total number of possible input (2^n) = 4
  - Maximum value ($2^n - 1$) = 3

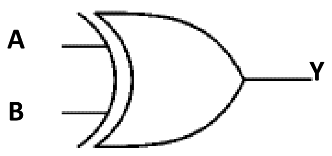
Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

- Boolean/Logic Expression

$$Y = \overline{A + B}$$

f) EXCLUSIVE-OR Gate (EX-OR)

- EX-OR gates can only have 2 inputs.
- This is like an OR gate but excluding both inputs being true.
- The output is true if inputs A and B are **DIFFERENT**.
- Standard logic symbol for NOR Gate



- Truth table for a 2-input NOR Gate
  - Total number of possible input (2^n) = 4
  - Maximum value ($2^n - 1$) = 3

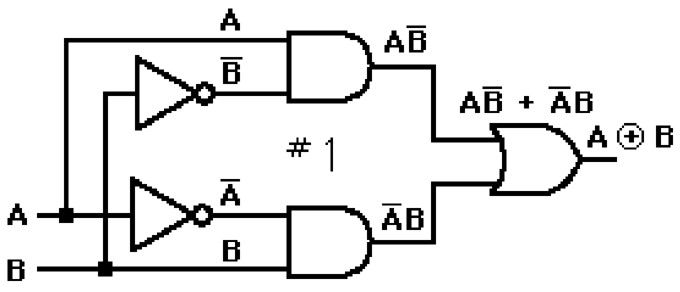
Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

– Boolean/Logic Expression

$$Y = A \oplus B$$

$$= A\overline{B} + \overline{A}B$$

– Logic Circuit



g) EXCLUSIVE-NOR Gate (EX-NOR)

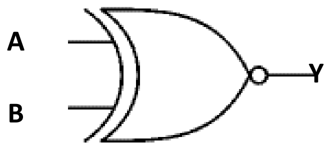
– This is an EX-OR gate with the output inverted, as shown by the 'o' on the output.

– EX-NOR gates can only have 2 inputs.

– The output is true if inputs A and B are the **SAME** (both true or both false)

– Standard logic symbol for NOR Gate

–



- Truth table for a 2-input NOR Gate
  - Total number of possible input (2^n) = 4
  - Maximum value ($2^n - 1$) = 3

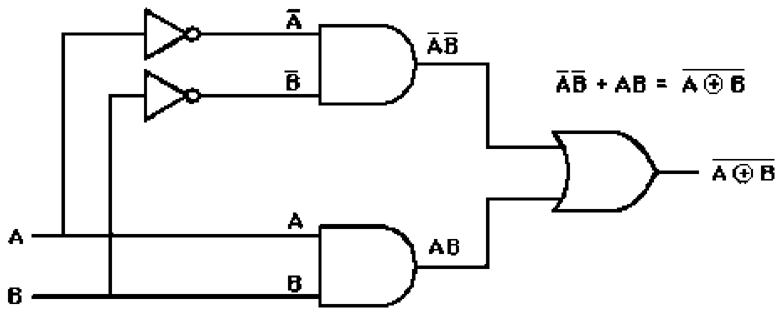
Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

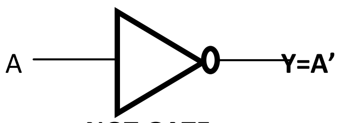
- Boolean/Logic Expression

$$Y = \overline{A \oplus B}$$

$$= \overline{A}\overline{B} + AB$$

- Logic Circuit

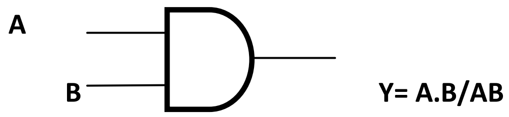

$\overline{A}$
$\overline{B}$
$\overline{A}\overline{B}$
$\overline{A}\overline{B} + AB = \overline{A \oplus B}$
$\overline{A \oplus B}$
A
B
AB
A
B



NOT GATE

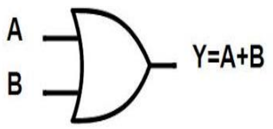
INPUT	OUTPUT
A	Y
0	1
1	0

Y= A'
A=0
Y=1



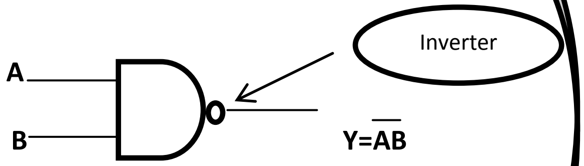
AND GATE

INPUT		OUTPUT
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



OR GATE

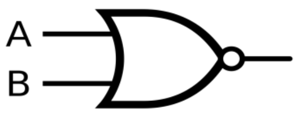
INPUT		OUTPUT
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



NAND GATE

INPUT		OUTPUT
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Inverter

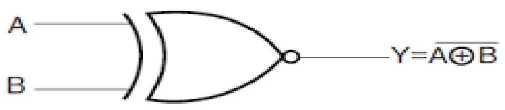


Y =$\overline{A+B}$

INPUT		OUPUT
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

NOR-GATE INPUT		OUTPUT
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

EX NOR GATE



$$Y = (\overline{A \oplus B}) = (A.B + \overline{A}.\overline{B})$$

INPUT		OUTPUT
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

If the input numbers are same, the output will be '1'.

If the input number are different, the output will be '0'.

INPUT		OUTPUT
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

EX-OR GATE

$= A\overline{B} + \overline{A}B$

If the input numbers are same, the output will be '0'.

If the input number are different, the output will be '1'.

Sequential Logic Circuit

- Sequential logic differs from combinational logic in that the output of the logic device is dependent not only on the present inputs to the device, but also on past inputs; *i.e.*, the output of a sequential logic device depends on its present internal state and the present inputs. This implies that a sequential logic device has some kind of *memory* of at least part of its ``history'' (*i.e.*, its previous inputs).
- The memory elements in a sequential circuit are called **flip-flops (FF)**. A flip-flop circuit has two outputs, one for the **normal value** and one for the **complement value** of the stored bit. Binary information can enter a flip-flop in a variety of ways and gives rise to different types of flip-flops. Flip-flops can be use as counter, register, memory devices and logic control circuits.

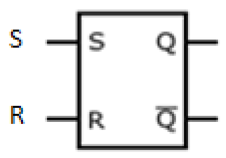
- 5 types of flip-flops are:
  a. SR Flip-Flop
  b. Clocked SR Flip-Flop
  c. JK Flip-Flop
  d. T Flip-Flop
  e. D Flip-Flop

SR Flip-Flop

- A flip-flop circuit can be constructed from two NAND gates (Active Low) or two NOR gates (Active High). Each flip-flop has two outputs, *Q* and *Q'*, and two inputs, *set* and *reset*. This type of flip-flop is referred to as an *SR flip-flop.*

a) SR NAND Flip-flop (Active Low)

Symbol



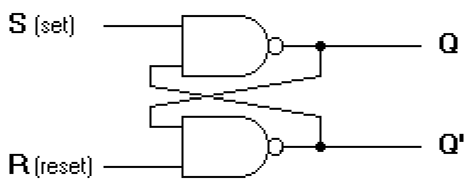
Truth Table

INPUT		OUTPUT	COMMENTS
S	R	Q	
0	0	0/1	Invalid
0	1	1	Set (1)
1	0	0	Reset (0)
1	1	0/1	No change

$Q = \overline{Q} = 1$

Note : Invalid (Q=Q=1)

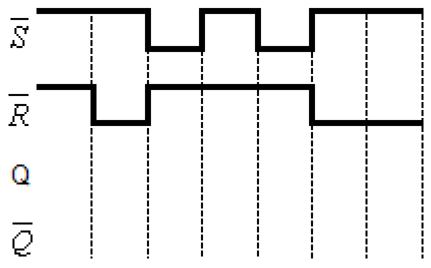
Logic Circuit



Timing Diagram

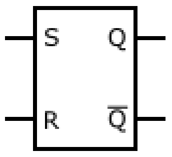
Example 1:

If $\overline{S}$ and $\overline{R}$ waveform are applied to the input below, determine the waveform that will be observed on the Q and $\overline{Q}$ output.



b) SR NOR Flip-flop (Active High)

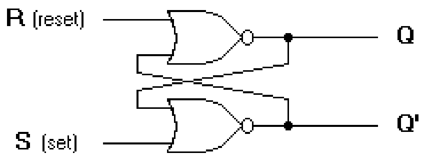
Symbol



Truth Table

INPUT		OUTPUT	COMMENTS
S	R	Q	
0	0	0/1	No change
0	1	0	Reset (0)
1	0	1	Set (1)
1	1	0/1	Invalid $\rightarrow \boxed{Q = \overline{Q} = 0}$

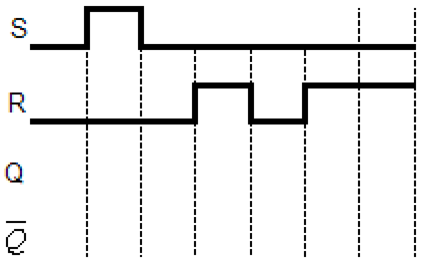
Logic Circuit



Timing Diagram

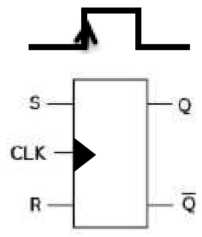
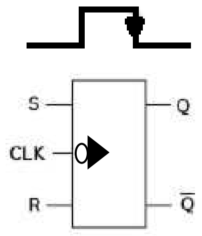
Example 1:

If *S* and R waveform are applied to the input below, determine the waveform that will be observed on the *Q* and $\overline{Q}$ output.

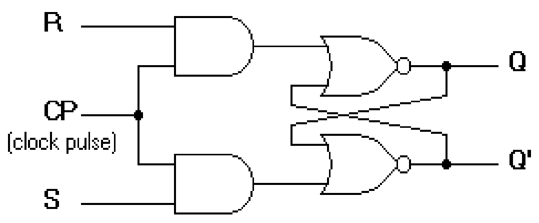


Clocked SR Flip-Flop

- Digital systems can operate either asynchronously and synchronously. In asynchronous systems, the outputs of logic circuits can change state any time one or more of the input change while in synchronous systems, the exact times at which any output can change states are determined by a signal commonly called the clock. The clock signal is distributed to all part of the systems, and most of the systems output can change state only when the clock makes transition.
- The triggering of a flip-flop is referring to the state of a flip-flop changed by a momentary change in the input signal. The basic circuits require an input trigger defined by a change in signal level. This level must be returned to its initial level before a second trigger is applied. Clocked flip-flops are triggered by pulses.
- 2 types of clock transition are:
  a. Positive edge trigger.
  b. Negative edge trigger.

Positive edge trigger	Negative edge trigger
✖ When the clock changes from a 0 to a 1. 	✖ When the clock changes from a 1 to a 0. 

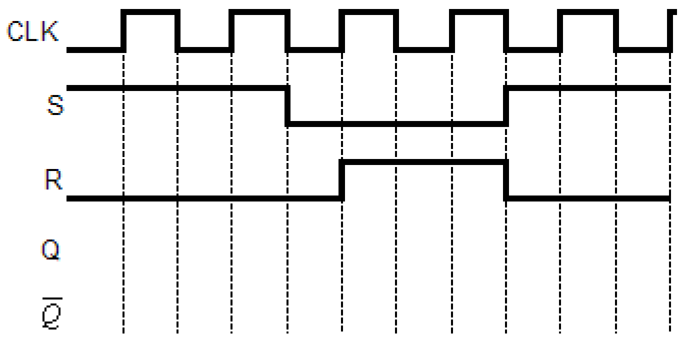
Logic Circuit



Timing Diagram

Example 1:

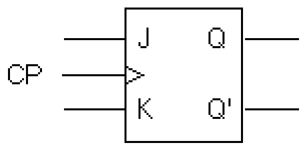
Determine the waveform that will be observed on the Q and $\overline{Q}$ output of this Active High Clocked SR flip-flop. Assume $Q_0 = 0$ and positive edge trigger clock.



JK Flip-Flop

- A JK flip-flop is a refinement of the SR flip-flop in that the invalid state of the SR type is defined in the JK type. Inputs J and K behave like inputs S and R to set and clear the flip-flop (note that in a JK flip-flop, the letter J is for set and the letter K is for clear). When logic 1 inputs are applied to both J and K simultaneously, the flip-flop switches to its complement state, ie., if Q=1, it switches to Q=0 and vice versa. This is also known as Toggle.

Symbol

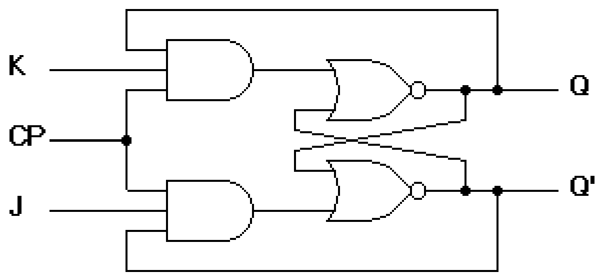


Truth Table

INPUT		OUTPUT	COMMENTS
J	K	Q	
0	0	0/1	No change
0	1	0	Reset (0)

1	0	1	Set (1)
1	1	0/1	Toggle

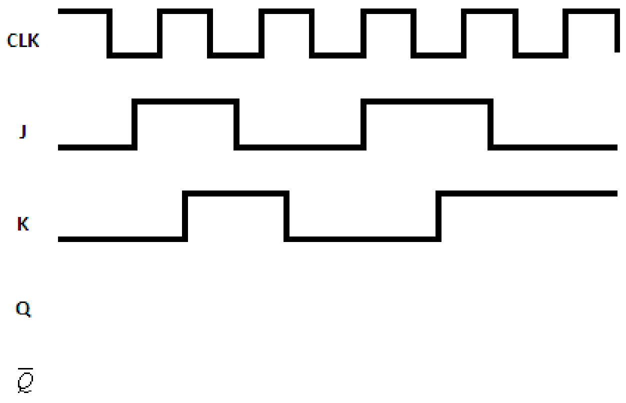
Logic Circuit



Timing Diagram

Example 1:

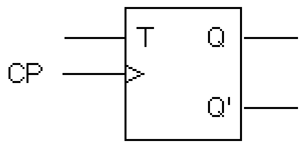
If J and K waveform are applied to the input below, determine the waveform that will be observed on the Q and $\overline{Q}$ output. Assume $Q_0 = 0$ and Negative edge trigger clock.



Flip-Flop (TOGGLE)

- The T flip-flop is a single input version of the JK flip-flop. The T flip-flop is obtained from the JK type if both inputs are tied together. The output of the T flip-flop "toggles" with each clock pulse.

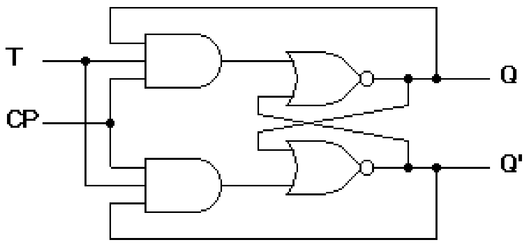
Symbol



Truth Table

INPUT	OUTPUT
T	Q
0	No change
1	Toggle

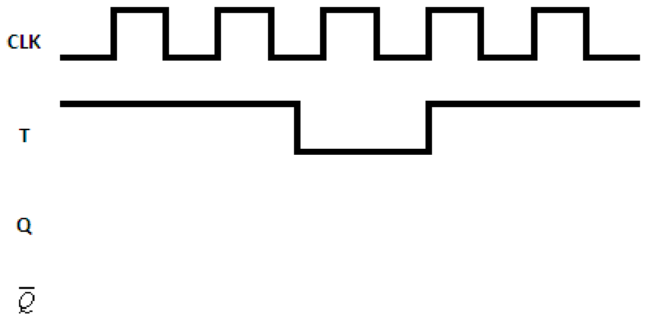
Logic Circuit



Timing Diagram

Example 1:

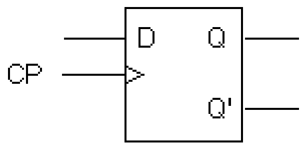
If T waveform is applied to the input below, determine the waveform that will be observed on the Q and $\overline{Q}$ output. Assume $Q_0 = 0$ and Positive edge trigger clock.



D Flip-Flop

- The D flip-flop is a modification of the clocked SR flip-flop. The D input is sampled during the occurrence of a clock pulse. If it is 1, the flip-flop is switched to the set state (unless it was already set). If it is 0, the flip-flop switches to the clear state.

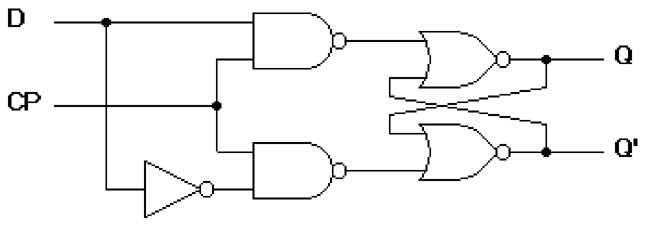
Symbol



Truth Table

INPUT		OUTPUT
D	CLOCK	Q
0	↑	0
1	↑	1

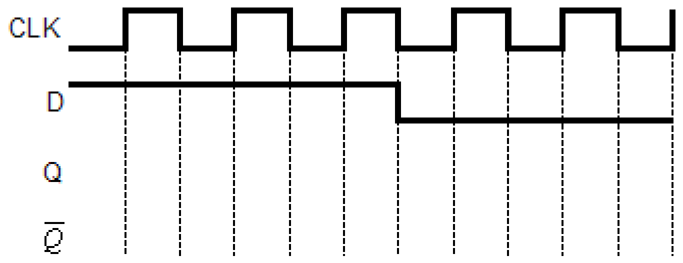
Logic Circuit



Timing Diagram

Example 1:

If D waveform is applied to the input below, determine the waveform that will be observed on the Q and $\overline{Q}$ output. Assume $Q_0 = 0$ and Positive edge trigger clock.



Registers

In a computer, a register is one of a small set of data holding places that are part of a computer processor . A register may hold a computer instruction , a storage address, or any kind of data (such as a bit sequence or individual characters). Some instructions specify registers as part of the instruction. For example, an instruction may specify that the contents of two defined registers be added together and then placed in a specified register. A register must be large enough to hold an instruction - for example, in a 32-bit instruction computer, a register must be 32 bits in length. In some computer designs, there are smaller registers - for example, half-registers - for shorter instructions. Depending on the processor design and language rules, registers may be numbered or have arbitrary names.

Categories of registers

Registers are normally measured by the number of bits they can hold, for example, an "8-bit register" or a "32-bit register". A processor often contains several kinds of registers, that can be classified accordingly to their content or instructions that operate on them:

- User-accessible registers – instructions that can be read or written by machine instructions. The most common division of user-accessible registers is into data registers and address registers.
    - **Data registers** can hold numeric values such as integer and, in some architectures, floating-point values, as well as characters, small bit arrays and other data. In some older and low end CPUs, a special data register, known as the accumulator, is used implicitly for many operations.

- **Address registers** hold addresses and are used by instructions that indirectly access primary memory.

Memory Organisation

Introduction To Computer Memory

In computing, memory refers to the physical devices used to store programs (sequences of instructions) or data (e.g. program state information) on a temporary or permanent basis for use in a computer or other digital electronic device.

Primary vs Secondary Memory Storage Devices

- **Primary memory** or the **main memory** is the memory that is directly accessed by the CPU to store and retrieve information. The primary memory itself is implemented by two types of memory technologies. The first is called Random Access Memory (RAM) and the other is read only memory (ROM)

- **Secondary memory** (mass memory/external memory/auxiliary memory) is a storage device that is not accessible directly by the CPU and used as a permanent storage device that retains data even after power is turned off. Hard drives, floppy disks, tapes, and optical disks are widely used for secondary storage.

INSTRUCTION SET AND ASSEMBLY LANGUAGE PROGRAMMING

Instruction and Instruction Set

An **instruction** is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called **Instruction Set** Eg. 8085 microprosessor has 246 instructions represented by 8 bit binary value. The 8 bit of binary value is called op-code or instruction byte.

Instruction set: the set of instructions that are interpreted directly in the hardware by the CPU. These instructions are encoded as bit strings in memory and are fetched and executed one by one by the processor. They perform primitive operations such as "add 2 to register i1", "store contents of o6 into memory location 0xFF32A228", etc. Instructions consist of an operation code (opcode) e.g., load, store, add, etc., and one or more operand addresses.

Computers with different microarchitectures can share a common instruction set
Eg. the Intel Pentium and the AMD Athlon implement nearly identical versions of the x86 instruction set, but have radically different internal designs

Classification of instruction set.

Instructions set can be classified into the following seven functional categories:

1. Data movement instructions
2. Compare instructions
3. Branch instructions
4. Arithmetic Instructions
5. Logic Instructions
6. Bit Manipulation Instructions
7. Shift and rotation
8. Stack and Subroutine Related Instructions

➢ Compare instructions

– All compare instructions subtract the source operand, usually the contents of one register (or memory location) from the contents of the destination operand, usually another register (or memory location) in order to set the CCR (except the X-bit). The results of the subtraction are discarded.

– Compare instructions include the following:

CMP	Source operand: Any of the addressing modes Destination: Must be a data register.
CMPA	Source operand: Any of the addressing modes Destination: Must be an address register.
CMPI	Source operand: An immediate value Destination: Any of the addressing modes except address register direct or immediate.
CMPM	Compares one memory location with another Only addressing mode permitted is address register indirect with auto-incrementing.

➢ Data movement instructions

– These instructions move data from one place to another.
– Data movement instructions include the following:

EXG	(EXchanGe) The contents of two registers will be exchanged
LEA	(Load Effective Address) Calculates a memory location and store it in an address register.
LINK	Allocates a stackframe.
MOVE	Copies the contents in one register/memory location to another register or another memory location.
MOVEA	(MOVE Address) Same as MOVE except that the destination is an address-register.
MOVEM	(MOVE Multiple) transfers many registers to or from the memory.
MOVEP	(MOVE Peripheral) transfers data to or from an 8 bits peripheral unit.
MOVEQ	(MOVE Quick) puts a constant in a dataregister.
PEA	(Push Effective Address) calculates a memory address and stores it on the stack.
SWAP	Swaps the word in a dataregister.

Information…
MOVE copies data from one location to another and may be qualified by ".B" to move 8 bits; ".W" to move 16 bits; and ".L" to move 32 bits.

MOVE does not change the source location only the destination location.

- **Branch instructions**

- Identified by the mnemonic Bcc where "cc" represents the condition to be checked.
- General form: Bcc Address_Label
- If the condition is true, then control will branch to "Address_Label".
- No effect on condition codes.

- These instructions can be grouped according the type of condition being checked:

  - Instructions that depend on a single CCR flag:
    BNE BEQ BPL BMI BCC BCS BVC BVS

  - Instructions for signed comparison:
    BGE BGT BLE BLT
  - Instructions for unsigned comparison:
    (BHS or BCC) BHI BLS (BLO or BCS)

- **Arithmetic Instructions**

- These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

- These instructions perform simple two complement operations on binary data.

ADD, ADDA, ADDI, ADDQ, ADDX	Different kinds of addition.
CLR	Clears an operand.
CMP, CMPA, CMPI, CMPM	Compares two operands
DVIS, DIVU	Integer division, signed and unsigned.
EXT	Makes a sign extension, byte to word or

	word to longword
MULS, MULU	Multiplication, signed and unsigned.
NEG, NEGX	Twocomplements a number
SUB, SUBA, SUBI, SUBQ, SUBQ	Different kinds of subtraction.
TAS	(Test And Set) used to synchronise more than one processor
TST	Compares an operand with 0.

- **Logic Instructions**

– These instructions perform various logical operations (AND, OR Exclusive-OR, Rotate, Compare and Complement) with the contents of the accumulator.

– These operations perform logical operations on binary numbers. A logical operation is either "true" (1) or "false" (0).

AND, ANDI Logical AND	on two binary integers
OR, ORI	Logical OR
EOR, EORI	Exclusive OR (XOR)
NOT	Returns the operans onecomplement (0 -> 1, 1 -> 0)

- **Bit Manipulation Instructions**

– These instructions affect single bits in a byte. All instructions test the bit before affecting it.

BTST	Tests a bit
BSET	Tests a bit, then set it (1)
BCLR	Tests a bit, then reset it (0)
BCHG	Tests a bit, then invert it (0 -> 1, 1 -> 0)

- **Stack and Subroutine Related Instructions**

– These instructions contain branches, jumps, calls.

Bcc	A group of 15 instruction that branches depending on the flags.
DBcc	15 instructions that perform loops.
Scc	16 instructions that will set/reset a byte depending on the flags.
BSR, JSR	Subroutine calls.

RTS	Return from a subroutine.
JMP	Absolute jumps.
RTR	Pops the PC and the flags from the stacks.

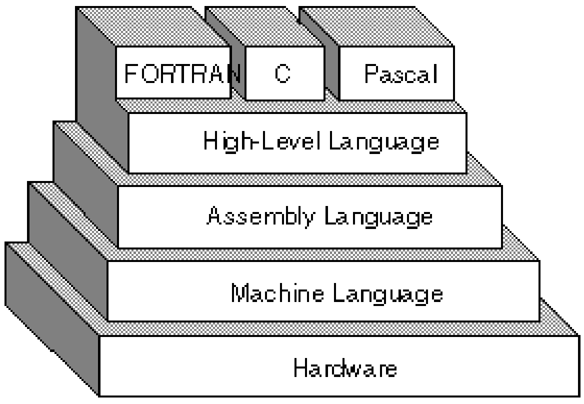
- **Shift and Rotation**

– These instructions perform arithmetical and logical shift and rotation with or without extra carry.

ASL, ASR	Arithmetic shift left resp right.
LSL, LSR	Locigal shift left resp right.
ROL, ROR	Rotation left resp right without extra carry.
ROXL, ROXR	Rotation left resp right through extra carry.

ASSEMBLY LANGUAGE

<u>Machine Language And Assembly Language</u>



Machine language

Machine languages are the only languages understood by computers. While easily understood by computers, machine languages are almost impossible for humans to use because they consist entirely of numbers.

Programmers, therefore, use either a high-level (C++, Java) programming language or an assembly language.

Assembly language

Programs written in high-level languages are translated into assembly language or machine language by a compiler. Assembly language programs are translated into machine language by a program called an assembler.

Low-level programming language for computers, microprocessors, microcontrollers, and other programmable devices. Implements a symbolic representation of the machine codes and other constants needed to program a given CPU architecture.

Used to translate assembly language statements into the target computer's machine code. Translation (a one-to-one mapping) from mnemonic statements into machine instructions and data.

The Addressing Modes Using Proper Instruction Format

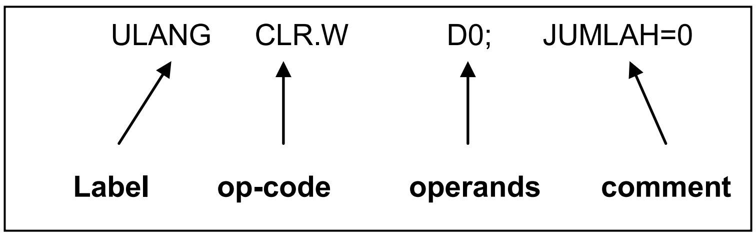
Instruction format

– Instructions are listed by mnemonic in alphabetical order. The information provided about each instruction is: its assembler syntax, its attributes (i.e., whether it takes a byte, word, or longword operand), its description in words, the effect its execution has on the condition codes, and the addressing modes it may take.

– The most common fields in instruction formats are:
  1. Mode field: Specifies the way the effective address is determined
  2. Operation code: Specifies the operations to be performed.
  3. Address field: Designates a memory address or a processor register

Mode	Opcode	**Address**

– Example of instruction format:



Opcodes and Operands

Two of the parts (OPCODE and OPERANDS) are mandatory. An instruction must have an OPCODE (the thing the instruction is to do), and the appropriate number of operands (the things it is supposed to do it to).

Labels

Symbolic names which are used to identify memory locations that are referred to explicitly in the program.

Comments

Messages intended only for human consumption. They have no effect on the translation process and indeed are not acted on by the Assembler. They are identified in the program by semicolons. The purpose of comments is to make the program more comprehensible to the human reader. They help explain a non intuitive aspect of an instruction or a set of instructions.

Addressing Modes

- Addressing modes are concerned with the way data is accessed.
- Addressing can be by actual address or based on a offset from a known position.
- Theoretically, only absolute addressing is required; however, other addressing modes are introduced in order to improve efficiency.

Types Of Addressing Modes

1. Absolute Addressing

Absolute Addressing uses the actual address of an operand; either a memory location (e.g: CLR.B $1234) or,
If a register is involved, this type is also called data register direct, e.g., MOVE.B D2,$2000

2. Immediate Addressing

With Immediate Addressing, the actual operand is part of the instruction;
Example: MOVE.B #25,D2

3. Immediate addressing with data registers

This is the simplest form of addressing. This mode is used to define a constant or set initial values of variables.
Advantage: no memory reference other than instruction fetch is required to obtain operand

Disadvantage: the size of the number is limited to the size of the address field, which most instruction sets is small compared to word length

Directly operate on the contents of a data register.
Example: MOVE.L D1, D0

4. Immediate addressing with address registers

Directly operate on the contents of an address register.
Example: MOVE.L A1, D0

5. Indirect addressing

Operate on the memory location pointed to by An.
Example: MOVE.L (A0), D0

6. Address Register Indirect Addressing

This addressing mode uses the 8 address registers. These registers are assumed to contain the address of the data rather than the data itself.
Example: CLR.B (A0)
Similar to indirect addressing. Address field of the instruction refers to a register.
The register contains the effective address of the operand.The address space is limited to the width of the registers available to store the effective address.

7. Address Register Indirect with Post-incrementing

A variation of address register indirect in which the operand address is incremented after the operation is performed.
The syntax is (Ai)+

8. Address Register Indirect with Pre-decrementing

A variation of address register indirect in which the operand is decremented before the operation is performed.
The syntax is -(Ai)

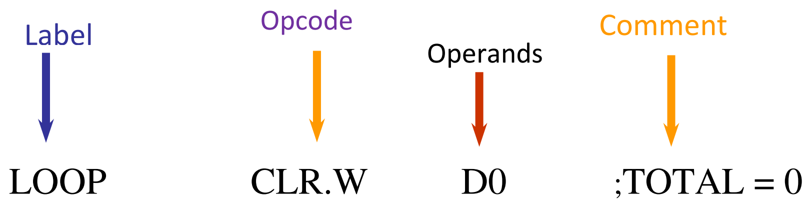
Assembly Language Instruction

- Series of statements which are either assembly language instructions or directives.
- Instructions are statements like ADD, SUB which are translated into machine code.
- Directives or pseudo-instructions are statements used by the programmer to direct the assembler on how to proceed in the assembly.

Statement format:

[label:] mnemonic [operands][;comments]

Example of instruction format:



Label:

- Cannot exceed 31 characters

Consists:

- Alphabetic characters both upper and lower case
- Digits 0 through 9
- comments start with semicolon, continue until end of line
- The first character cannot be a digit
- One instruction per line of code
- Spacing: at least one space required after each instruction (mnemonic or pseudo-op), otherwise doesn't matter and last line of program must be END pseudo-op.

Label:

- Must end with a colon when it refers to an opcode generating instruction. Do not need to end with a colon when it refers to a directive.

Mnemonic and operands:

- Instructions are translated into machine code. Directives do not generate machine code. They are used by the assembler to organize the program and direct the assembly process.

Comments:

- Comment must begin with a "**;**" and its ignored by the assembler.
- Comment should be on a line by itself or at the end of a line:
- Eg: ;My first comment
  MOV AX,1234H ;Initializing….
- Indispensable to the programmers because they make it easier for someone to read and understand the program

General pseudo-op

ORG

- The function of ORG (origin) is to set an address of instruction or data.

The format:	ORG	address
Example:	ORG	$2000

EQU

- The function of EQU (equate) is to give a value for certain symbol.

The format:	Symbol	EQU	value
Example:	SIZE	EQU	20

DC

- The function of DC (define constant) is to fill in certain values in a memory.

The format:	[label]	DC.data_size	value
Example:		ORG	$2000

DC.W	3
DC.B	$23,49
DC.L	10
DC.W	1,4,9,16

002000	00	03	DC.W 3
002002	23	49	DC.B $23,49
002004	00	00	DC.L 10
002006	00	0A	
002008	00	01	DC.W 1,4,9,16
00200A	00	04	
00200C	00	09	
00200E	00	10	

Figure 2: A memory location of the sequence above

DS

- The function of DS (define storage) is almost like to DC command. However it will not fill any information to the memory.
- The format: [label] DC.data_size value
- Example:

ARRAY	DS.W	1
STRING	DS.B	8
PTR	DS.L	1

002000	00	00	DS.W 1
002002	00	00	DS.B 8
002004	00	00	
002006	00	00	
002008	00	00	
00200A	00	00	DS.L 1
00200C	00	00	

Figure 3: A memory location of the sequence above

END

- The function of END is to tell the assembler that the program has ended.
- The format: END [label]

Simple program in assembly language program

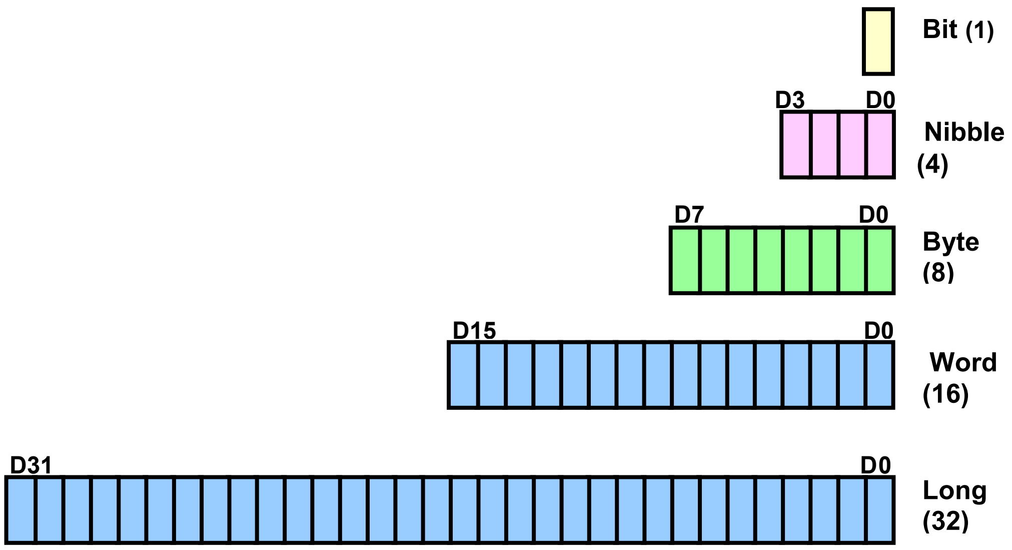
Data sizes

68000 instructions can direct the processor to work on five data types:

a) bit
b) Binary Coded Decimal (BCD - 4 bits)
c) Byte (8 bits)
d) Word (16 bits)
e) Longword (32 bits)

Note:
$ = value for hexadecimal
@ = value for octal
% = value for binary
& or blank = decimal
'AB' = character ASCII

- Bit:
  - ✓ Most basic representation.
  - ✓ Contains either 0 or 1.
  - ✓ Can be grouped together to represent more meaning.
- Nibble: 4 bits.
  - ✓ Can represent 16 values (24).
  - ✓ Not recognized in M68k.
  - ✓ Need to write special program to handle.
- Byte: 8 bits.
  - ✓ Indicated by ".B" notation.
  - ✓ Can hold value up to 256 (28).
- Word: 16 bits.
  - ✓ Length of most instructions in M68k.
  - ✓ Can hold value up to 65,535 (216).
  - ✓ Indicated by ".W" notation.
- Long Word: 32 bits.
  - ✓ Length of data registers in M68k.
  - ✓ Can hold value up to 4,294,967,296 (232).
  - ✓ Indicated by ".L" notation.



Data Register Direct

MOVE.B D0,D3

Before

REGISTER	CONTENTS
D0	10204FFF
D3	1034888A

After

REGISTER	CONTENTS
D0	10204FFF
D3	103488FF

Only bit 0-7 involved in this case because this operation only involved in byte

MOVE.W D0,D3

Before

REGISTER	CONTENTS
D0	10204FFF
D3	1034888A

After

REGISTER	CONTENTS
D0	10204FFF
D3	10344FFF

Only bit 0 – 15 involved in this case because this operation only involved in word.

MOVE.L D0,D3

Before

REGISTER	CONTENTS
D0	10204FFF
D3	1034888A

After

REGISTER	CONTENTS
D0	10204FFF
D3	10204FFF

Only bit 0 – 31 involved in this case because this operation only involved in long word.

Address Register Direct

MOVEA.L A3,A0

Before

REGISTER	CONTENTS
A0	00200000
A3	0004F88A

After

REGISTER	CONTENTS
A0	0004F88A
A3	0004F88A

32 bits involve because this operation is involve in long word.

MOVEA.W A3,A0

Before

REGISTER	CONTENTS
A0	00200000
A3	0004F88A

After

REGISTER	CONTENTS
A0	FFFFF88A
A3	0004F88A

16 bits involve because this operation is involved in word but the source operand for MSB =1 so the product is used sign-extended

Absolute Long Mode

CLR.B $10000

010000	42	51
010002	55	13

→

010000	**00**	51
010002	55	13

CLR.W $10000

010000	42	51
010002	55	13

→

010000	**00**	**00**
010002	55	13

CLR.L $10000

010000	42	51
010002	55	13

→

010000	**00**	**00**
010002	**00**	**00**

Immediate

MOVE.L #$1FFFF, D0

Before:D0 = 12345678

After: D0 = 0001FFFF

Quick Immediate

1. **MOVEQ #$2C, D3**

Before: D3 = 1234562C

After: D3 = 0000002C

2. MOVEQ #$8F, D3

Before: D3 = 1234568F

After: D3 = FFFFFF8F

Arithmetic operation

The basic arithmetic operations are **addition (+)**, **subtraction (-)**, **multiplication (x)** and **division (/)**

Example 1: **ADD.B D0, D1**
Before : D0 = 00000011 , D1 = 00000022
After : D0 = 00000011 , D1 = 00000033

Example 2: **ADD.W #$A2,D1**
Before : D1 = 00100500
After : D1 = 001005A2

Example 3 :
ADD.L D0,D1
Before : D0 = 10001111 , D1 = 00002222
After : D0 = 00001111 , D1 = 10003333

Example 4 : **ADD.W $1000,D1**
Before : D1 = 00110051
After : D1 = 00110081

$1000	30	31
$1002	33	34
$1004	35	36

Example 5: **SUB.W #$80,D3**
Before : D3 = $001122AB
After : D3 = $0011222B

Example 6: **SUB.W D0, D1**
Before : D0 = 00001111 , D1 = 00002222
After : D0 = 00000011 , D1 = 00001111

Logic operation

The logic operations are **OR**, **AND** and **NOT.**

Example 1 : AND.B #$3E,D1
Before : D1 = $12345674
After : D1 = $12345634

Example 2: OR.B D0,D1
Before : D1 = $1234563E , D0 = $98765474
After : D1 = $1234567E , D0 = $98765474

Simple Program

```
ORG   $1000

MOVE.B      #$1, D0
MOVE.B      #$2, D1
MOVE.B      #@3, D2

SUB    D0, D1
SUB    D1,D2

END
```

OUTPUT

```
MOVE.B      #$1, D0   ⇨     D0 = 00000001
MOVE.B      #$2, D1   ⇨     D1 =  00000002
MOVE.B      #@3, D2   ⇨      D2 = 00000003

SUB    D0, D1       ⇨      D1 = 00000001
SUB    D1,D2        ⇨      D2 = 00000002
```

Below show the calculation doing by assembly program using arithmetic and logic operation, you can solve the operation below with another way. This is how to write the assembly language program by using 68000 Motorola's processor.

$(227_8 + ABC_{16})$ OR 10011_2

Answer:

```
MOVE.W       #@227, D0

MOVE.W       #$ABC, D1

ADD.W        D0, D1

MOVE.W       #%10011, D2

OR.W         D1, D2
```

Exercise 1:

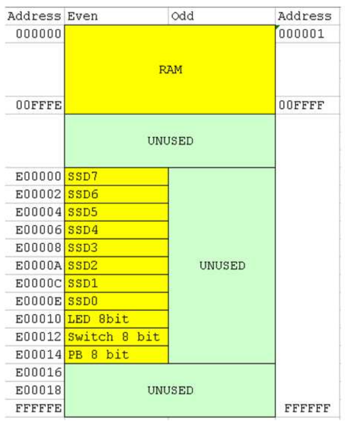
Write a program to add the content 200, 202 AND 204 where each variable A, B and C. assume that the program starts at address $2000.

Exercise 2:

Write a program to add together two 8-bit numbers stored in the memory locations called NILAI1 and NILAI2, and stores the sum in the memory location called KEPUTUSAN using an assembly language. Assume value for NILAI1 is 100 and NILAI2 is 200.

Introduction to EASy68K Cross Assembler and Simulator

- EASy68K has an editor, an assembler and an simulator for the Motorola 68000 CPU that run on Windows Operating System and Intel architecture.
- It is used to write and executes programs for the Motorola M68k architecture without extra hardware



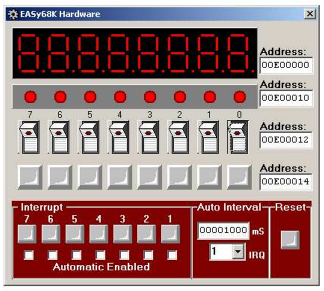
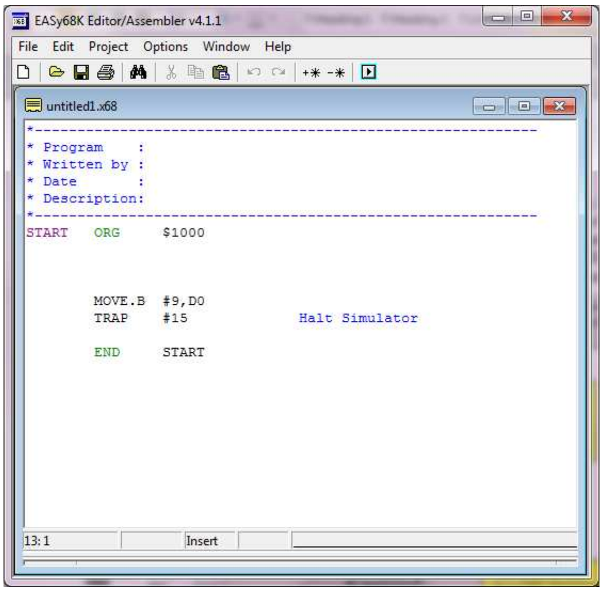


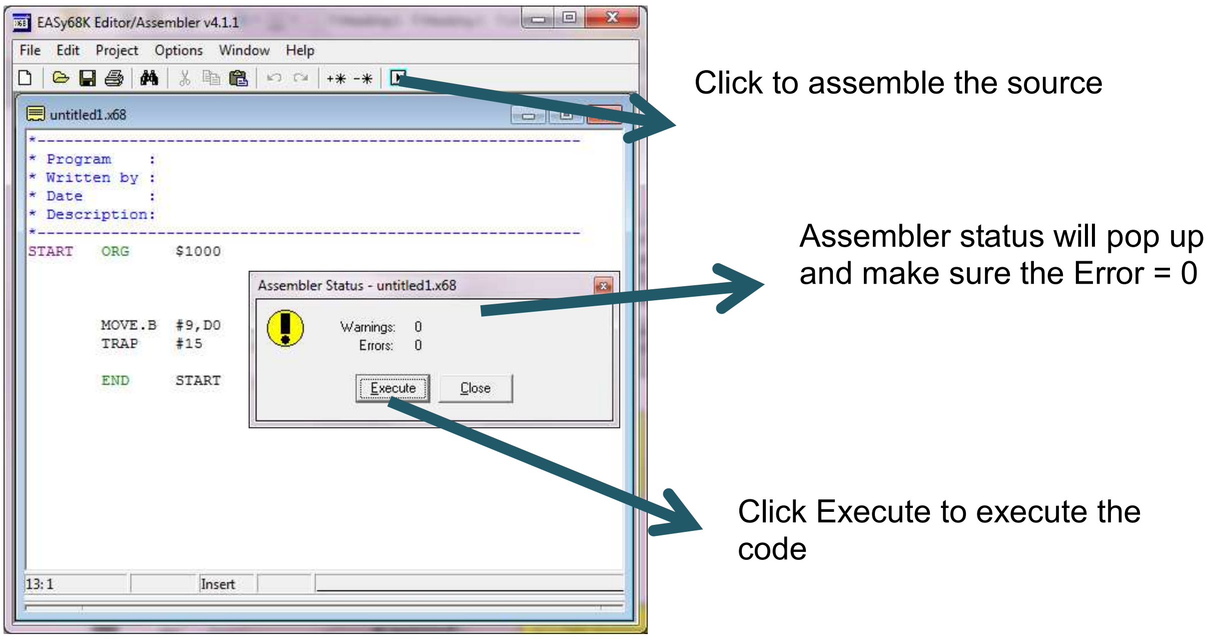
Figure 1(a) Figure 1(b).

- The simulator emulates a Motorola M68000 microprocessor system with the default memory map shown Figure 1(a) and simulate the hardware shown in Figure 1(b). The memory map of the hardware can be changed by the user.

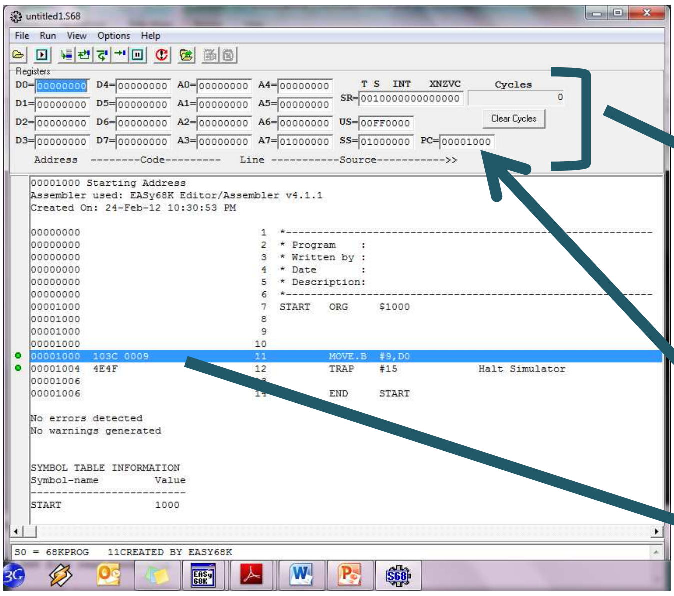

EASy68K Editor/Assembler v4.1.1
File Edit Project Options Window Help
untitled1.x68
13: 1
Insert

```
*-----------------------------------------------------------
* Program    :
* Written by :
* Date       :
* Description:
*-----------------------------------------------------------
START   ORG     $1000


        MOVE.B  #9,D0
        TRAP    #15             Halt Simulator

        END     START
```

- This code puts your program in memory at 100016.
- The END assembler directive has the label 'START' which point to the beginning of the program (eg: at address $1000).
- You must not put data here (i.e: immediately after the ORG statement!
- The first line after START must be a valid 68K instruction.


EASy68K Editor/Assembler v4.1.1
File Edit Project Options Window Help
untitled1.x68
Assembler Status - untitled1.x68
Warnings: 0
Errors: 0
Execute
Close
13: 1
Insert
Click to assemble the source
Assembler status will pop up and make sure the Error = 0
Click Execute to execute the code

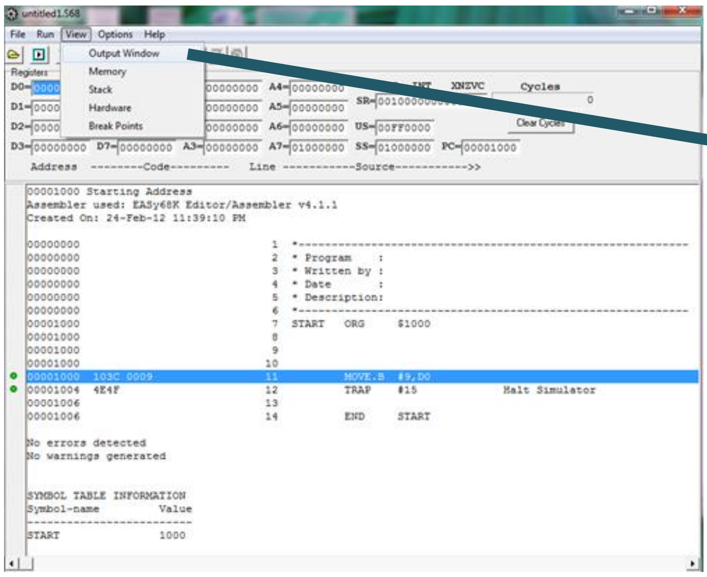
```
*-----------------------------------------------------------
* Program    :
* Written by :
* Date       :
* Description:
*-----------------------------------------------------------
START   ORG     $1000


        MOVE.B  #9,D0
        TRAP    #15

        END     START
```



```
D0=00000000 D4=00000000 A0=00000000 A4=00000000      T S  INT  XNZVC      Cycles
D1=00000000 D5=00000000 A1=00000000 A5=00000000  SR=0010000000000000           0
D2=00000000 D6=00000000 A2=00000000 A6=00000000  US=00FF0000
D3=00000000 D7=00000000 A3=00000000 A7=01000000  SS=01000000  PC=00001000
   Address  --------Code---------   Line -----------Source----------->>

00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v4.1.1
Created On: 24-Feb-12 10:30:53 PM

00000000                            1  *-----------------------------------------------------------
00000000                            2  * Program    :
00000000                            3  * Written by :
00000000                            4  * Date       :
00000000                            5  * Description:
00000000                            6  *-----------------------------------------------------------
00001000                            7  START    ORG     $1000
00001000                            8
00001000                            9
00001000                           10
00001000  103C 0009                11           MOVE.B  #9,D0
00001004  4E4F                     12           TRAP    #15             Halt Simulator
00001006                           13
00001006                           14           END     START

No errors detected
No warnings generated


SYMBOL TABLE INFORMATION
Symbol-name          Value
-------------------------
START                1000

S0 = 68KPROG   11CREATED BY EASY68K
```

The simulated register

The first line of the code pointed by **PC** declared by the END START directive



```
D0=0000                          00000000 A4=00000000      INT  XNZVC      Cycles
D1=0000                          00000000 A5=00000000  SR=0010000                0
D2=0000                          00000000 A6=00000000  US=00FF0000
D3=00000000 D7=00000000 A3=00000000 A7=01000000  SS=01000000  PC=00001000
   Address  --------Code---------   Line -----------Source----------->>

00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v4.1.1
Created On: 24-Feb-12 11:39:10 PM

00000000                            1  *-----------------------------------------------------------
00000000                            2  * Program    :
00000000                            3  * Written by :
00000000                            4  * Date       :
00000000                            5  * Description:
00000000                            6  *-----------------------------------------------------------
00001000                            7  START    ORG     $1000
00001000                            8
00001000                            9
00001000                           10
00001000  103C 0009                11           MOVE.B  #9,D0
00001004  4E4F                     12           TRAP    #15             Halt Simulator
00001006                           13
00001006                           14           END     START

No errors detected
No warnings generated


SYMBOL TABLE INFORMATION
Symbol-name          Value
-------------------------
START                1000
```

To see what your program has displayed, select the View menu and then click "Output Window"

THE CENTRAL PROCESSING UNIT

Introduction of Central Processing Unit (CPU)

- The part of computer that performs the bulk of data processing operations is called the central processing unit (CPU).
- The CPU is commonly referred to the 'brains of a computer"
- Also know as processor
- Responsible for executing a sequence of instructions called a program
- The program will take **inputs**, **process** and **output** the results.

- The CPU is made up of three major parts; Control unit, Register set and Arithmetic Logic Unit (ALU) as shown in Figure 1.

  - Register set – stores the most frequently used instructions and data
  - Arithmetic Logic Unit (ALU) – performs arithmetic or logical operations
  - Control – Coordinates and controls all parts of the computer system

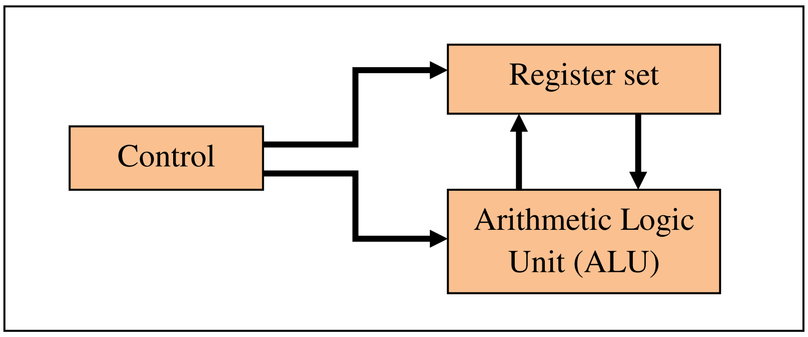


Figure 1: Major components of CPU

Instruction cycle

The time period during which one instruction is fetch from memory and execute when a computer given an instruction in machine language. Each instruction is further divided into sequence of phases. After the execution the program counter is incremented to point to the next instruction.

Phase of cycle : - Fetch cycle
- Decode cycle
- Execute cycle

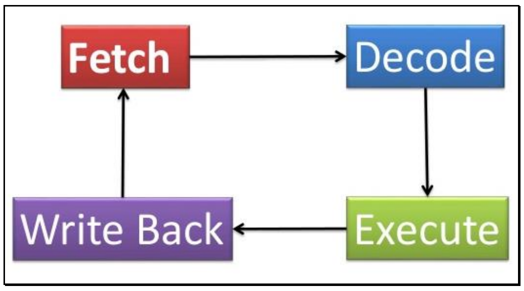


Figure 2: Instruction cycle

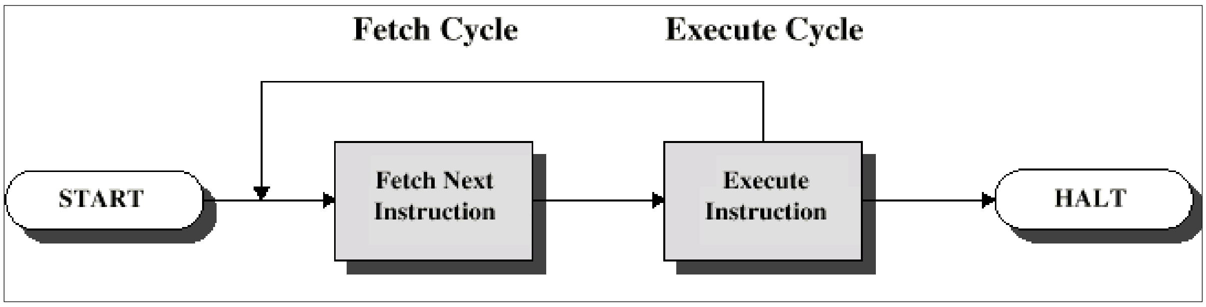


Figure 3: Fetch and execute cycle

Fetch Cycle

- Takes the address required from memory, stored it in the instruction register and moves the program counter
- Program Counter (PC) holds address of next instruction to fetch
- Processor fetches instruction from memory location pointed to by PC

Decode Cycle

- Figure out what the program is telling the computer to do
- Here, the control unit checks the instruction that is now stored within the instruction register
- It determines which opcode and addressing mode have been used and as such what actions need to be carried out in order to execute the instruction in question

Execute Cycle

- Perform the requested action
- The actual actions which occur during the execute cycle of an instruction depend on both the instruction itself and the addressing mode specified to be used to access the data that may be required

Four categories of actions

1. Processor-memory
   - data transfer between CPU and main memory

2. Processor I/O
   - Data transfer between CPU and I/O module

3. Data processing
   - Some arithmetic or logical operation on data

4. Control
   - Alteration of sequence of operations (e.g. jump)
   - Instruction execution may involve a combination of these

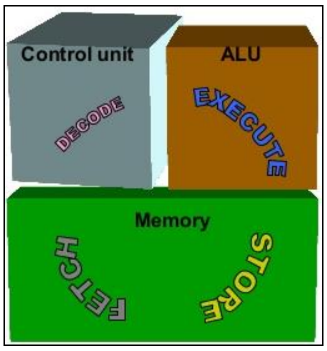


Figure 4 : How a CPU works – Fetch Execute Cycle

Basic Organization of Stack in Computer System

Stack

– 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. It is useful for compound arithmetic operations and nested subroutine calls.

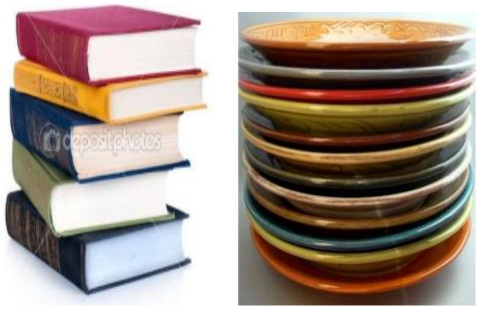


Figure 5: Examples of stack

– The stack in digital computers is a group of memory locations with a register that holds the address of top of element. This register that holds the address of top of element of the stack is called ***Stack Pointer***.

– **Stack Operations**
The two operations of a stack are:
1. **Push:** Inserts an item on top of stack.
2. **Pop :** Deletes an item from top of stack.

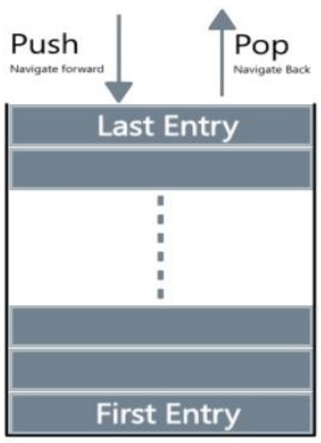


Figure 6 : Operation of stack

– **Implementation of Stack**

  In digital computers, stack can be implemented in two ways:

  1.Register Stack

  2.Memory Stack

– **Register Stack**

  - A stack can be organized as a collection of finite number of registers that are used to store temporary information during the execution of a program. The stack pointer (SP) is a register that holds the address of top of element of the stack.

– **Memory Stack**

  - A stack can be implemented in a random access memory (RAM) attached to a CPU. 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. The starting memory location of the stack is specified by the processor register as *stack pointer*.

<u>Reverse Polish Notation</u>

Infix notation: - The general way of writing arithmetic expressions is known as infix Notation
- Operators are written between two operands
- <operand> <operator> <operand>
- Eg: A + B

Postfix notation: - Also known as **Reverse Polish Notation**
- Operators are written after the operands
- *<operand> <operand> <operator>*
- eg: AB+

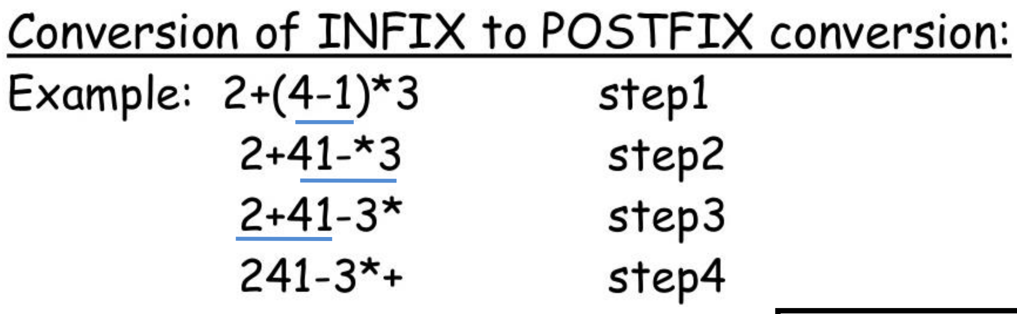
Prefix notation : - Also known as Polish Notation
- Operators are written before the operands
- <operator> <operand> <operand>
- eg: +AB

– **Reverse polish notation** : is a **postfix notation** (places operators after operands)

– **(Example)**

Infix notation	**D + E**	
Reverse Polish notation	**DE+**	also called postfix.

– A stack organization is very effective for evaluating arithmetic expressions
  - A * B + C * D → (AB *)+(CD *) → **AB * CD * +**
  - (3 * 4) + (5 * 6) → **34 * 56 * +**



Conversion of INFIX to POSTFIX conversion:

Example: 2+(4-1)*3 step1
2+41-*3 step2
2+41-3* step3
241-3*+ step4

Exercise :

i) (x + y) - z
ii) w * ((x + y) - z)
iii) (2 * a) / ((a + b) * (a - c))

– **Evaluation**

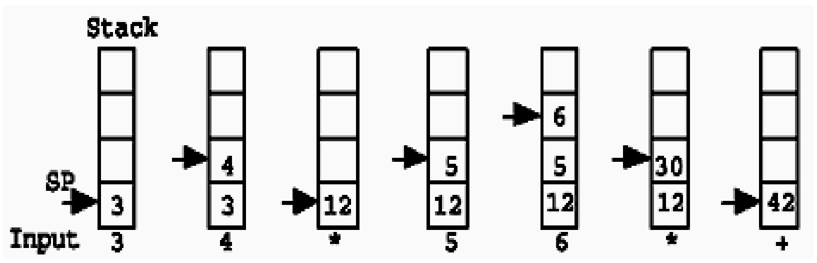
Evaluation procedure:

1. Scan the expression from left to right.
2. When an operator is reached, perform the operation with the two operands found on the left side of the operator.
   Replace the two operands and the operator by the result obtained from the operation.
3. (Example)
   infix ***3 * 4 + 5 * 6 = 42***
   postfix ***3 4 * 5 6 * +***
   12 5 6 * +
   12 30 +
   42

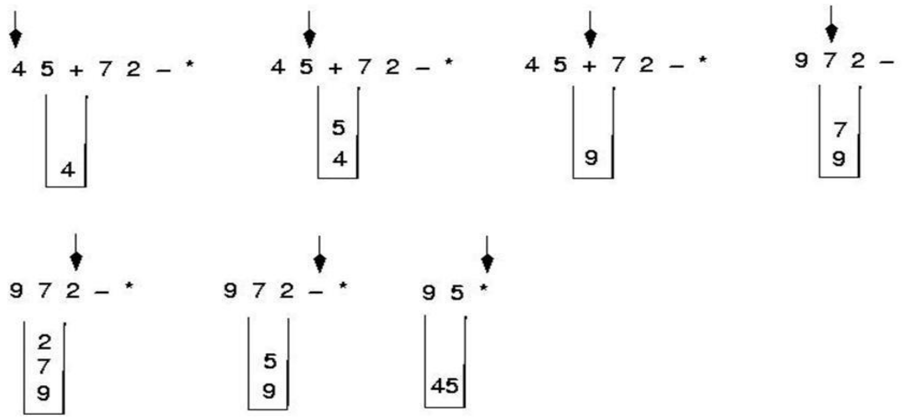
– Reverse Polish notation evaluation with a stack. Stack is the most efficient way for evaluating arithmetic expressions.

Stack evaluation:
Get value
If value is data: push data
Else if value is operation: pop, pop evaluate and push.

– **(Example) using stacks to do this.**
  i) ***3 * 4 + 5 * 6 = 42***
  => 3 4 * 5 6 * +



ii) **(4 + 5) (7 – 2)**
=> 4 5 + 7 2 - *



Reduced Instruction Set Computer (RISC) and Complex Instruction Set Computers (CISC)

Complex Instruction Set Computers (CISC) has a large instruction set, with hardware support for a wide variety of operations. In scientific, engineering, and mathematical operations with hand coded assembly language (and some business applications with hand coded assembly language), CISC processors usually perform the most work in the shortest time.
Reduced Instruction Set Computers (RISC) has a small, compact instruction set. In most business applications and in programs created by compilers from high level language source, RISC processors usually perform the most work in the shortest time.

RISC architecture

The first prototype computer to use reduced instruction set computer (RISC) architecture was designed by IBM researcher John Cocke and his team in the late 1970s. For his efforts, Cocke received the Turing Award in 1987, the US National Medal of Science in 1994, and the US National Medal of Technology in 1991.

IBM RT PC

The RISC Technology Personal Computer (RT PC) was introduced in 1986, and featured the 32-bit RISC architecture.

IBM RS/6000

The IBM RS/6000 was released in 1990. It was the first machine to feature the IBM POWER architecture. The RS/6000 has gone through several name changes throughout the years, including IBM eServer™ pSeries®, IBM System p® and IBM Power Systems™.

The differences between RISC and CISC

Reduced Instruction Set Computer (RISC)	**Complex Instruction Set Computer (CISC)**
• Software	• hardware
• Single clock, reduce instruction	• Multi clock complex instruction
• Single word instruction	• Variable length instruction
• Simple operations	• Complex operations
• Register to register. “LOAD” and “STORE” are independent instruction	• Memory-to-Memory. “LOAD” and “STORE” incorporated in instructions
• Low cycles per second, large code size	• Small code sizes, high cycles per second
• Spend more transistors on memory registers	• Transistors used for storing complex instructions

Concept of Pipelining

A technique used in advanced microprocessors where the microprocessor begins executing a second instruction before the first has been completed (instruction pre-fetch). That is, several instructions are in the pipeline simultaneously, each at a different processing stage.

The pipeline is divided into segments and each segment can execute its operation concurrently with the other segments. When a segment completes an operation, it passes the result to the next segment in the pipeline and fetches the next operation from the preceding segment. The final results of each instruction emerge at the end of the pipeline in rapid succession.

Although formerly a feature only of high-performance and RISC -based microprocessors, pipelining is now common in microprocessors used in personal computers. Intel's Pentium chip, for example, uses pipelining to execute as many as six instructions simultaneously.

Pipelining is also called pipeline processing.

COMPUTER SYSTEM ARCHITECTURE course is designed to introduce the basic concepts on which the stored program digital computer is formulated. These include the introduction of computer architecture and computer organisation, and the representation and manipulation of numbering system. This goal addresses the question on how does a computer work and how it is organized. The course also provides students with foundation knowledge of the Central Processing Unit and assembly language programming.

Murniyati Binti Abdul is a Lecturer in the Department of Information and Communication Technology at Polytechnic Sultan Mizan Zainal Abidin, Terengganu.

MURNIYATI BINTI ABDUL II COMPUTER SYSTEM ARCHITECTURE

e ISBN 978-967-0047-14-0

9 789670 047140

COMPUTER SYSTEM ARCHITECTURE