REPUBLIC OF THE PHILIPPINES
POLYTECHNIC UNIVERSITY OF THE PHILIPPINES
STA. MESA, MANILA
COLLEGE OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

CMPE 30224

Computer Architecture and Organization (Lecture)

INSTRUCTIONAL MATERIAL

ENGR. ROLITO L. MAHAGUAY, MSE-CpE, PCpE

Table of Contents

CHAPTER 1 - COMPUTER ARCHITECTURE AND ORGANIATION

Course Overview:

This course includes the study of the evolution of computer architecture and the factors influencing the design of hardware and software elements of computer systems. The focus is on the understanding of the design issues specifically the instruction set architecture and hardware architecture.

INTRODUCTION TO COMPUTER ARCHITECTURE AND ORGANIZATION

Overview:

This chapter is designed to let the students understand the Structure and Function of Computer architecture and organization, designing for performance, computer components, computer functions, interconnection structures and bus interconnection.

Note: Always check the attendance of the class before the start of each session.

Learning Objectives:

After successful completion of this lesson, you should be able to:

- Understand the components that comprise a computer and their interconnections. Sets of instructions, called programs.
- Describe the computations that computers carry out. The instructions are strings of binary digits.
- Discuss how computer interprets the instructions and send signals to other components that cause the instruction to be carried out

Course Materials:

- Computer
- Files (INTRODUCTION TO COMPUTER ARCHITECTURE AND ORGANIZATION)
- Demonstration Materials
    - Handouts
    - PowerPoint Presentation
    - Other Instructional materials

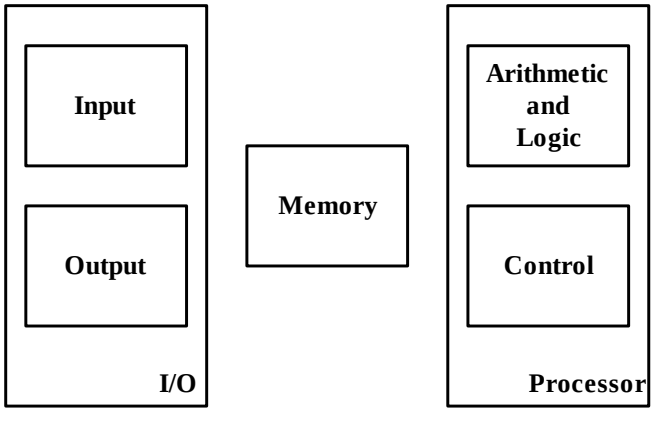
BASIC STRUCTURE OF COMPUTER HARDWARE AND SOFTWARE

INTRODUCTION

- A ***digital computer*** is a fast electronic calculating machine that accepts digitized input information, processes it according to a list of internally stored instructions, and produces the resulting output information.
- The list of instructions is called a computer ***program***, and the internal storage is called computer ***memory***.
- Types of Computers

1. ***Personal Computers***. This is the most common computer which has found wide use in homes, schools, and business offices.
2. ***Workstations***. Although still of desktop dimensions, these machines have a computational power which is significantly higher than that of personal computers. Workstations are often used in engineering applications, especially for interactive design work (CAD/CAM).
3. ***Mainframes***. A large and powerful computer used for business data processing in medium to large corporations that require much more computing and storage capacity than workstations can handle.
4. ***Supercomputers***. These are used for large-scale numerical calculations found in applications such as weather forecasting and aircraft design and simulation. In mainframes and supercomputers, the main functional units may comprise a number of separate and often large parts

- In its simplest form, a computer consists of five functionally independent main parts: ***input***, ***memory***, ***arithmetic and logic***, ***output***, and ***control*** units.



BASIC FUNCTIONAL UNITS OF A COMPUTER

☞ The ***Input Unit*** accepts coded information from human operators or from other computers.

Examples:

keyboard, joystick, mouse, trackball, scanner, bar code readers

☞ ***Memory Unit***

1. ***Primary Storage*** or ***Main Memory***. This is where programs are stored during their execution. The MM is a fast memory capable of operating at electronic speeds.
   - The information in MM is often processed in groups of fixed size called ***words***. The number of bits in a word is the ***word length*** of the computer.
   - Typical word lengths range from 16 to 64 bits. The MM is organized so that the contents of one word, containing *n* bits, can be stored or retrieved in one basic operation.
   - Since programs must reside in MM during execution, MM is often referred to as the *bottleneck* in most computer operations.
   - To provide easy access to any word in MM, a distinct address is associated with each word location. Addresses are numbers that identify successive locations.
   - MM is also known as ***random-access memory*** (RAM). RAM is memory in which any location can be reached in a short, fixed amount of time.
   - The time required to access one word is the ***memory access time***. For RAMs, this time is fixed, independent of the location of the word being accesses. It typically ranges from 10 to 100 nanoseconds for most modern computers.
2. ***Secondary Storage***. This is used when large amounts of data have to be stored, particularly if some of the data need not be accessed very frequently.

Examples: magnetic disks, drums, tapes

☞ ***Arithmetic and Logic Unit*** (ALU). This is where the execution of most operations takes place. It contains a number of high-speed storage elements called ***registers***.

- used for temporary storage of frequently used operands.
- Each register can store one word of data.
- access times are 5 to 10 times faster than memory access times.

☞ ***Control Unit*** (CU). This is the nerve center of a computer. It sends control signals to other units and senses their state.

The ALU together with CU is the ***Central Processing Unit*** (CPU) of a computer. The control and arithmetic and logic units are usually many times faster than other devices connected to a computer system. This enables a single processor to control a number of external devices such as video terminals, magnetic tape and disk memories, sensors, displays, and mechanical controllers. This is possible because the vast difference in speed enables the fast processor to organize and control the activity of many slower devices

☞ ***Output Unit***. It sends processed results to the outside world.

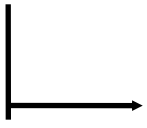
Examples: CRT/monitor, printers, plotters, modems

☞ The operation of a computer can be summarized as follows:

1. The computer accepts information in the form of programs and data through an input unit and stores it in memory.
2. Information stored in the memory is fetched, under program control, into an arithmetic and logic unit, where it is processed.
3. Processed information leaves the computer through an output unit.
4. All activities inside the machine are directed by the control unit.

BASIC CONCEPTS OF COMPUTER ARCHITECTURE

- ***Computer Architecture*** is the design of computers, including their instruction sets, hardware components, and system organization.
- Most computers follow the ***Von Neumann Architecture***.



Is also known as the ***Stored Program Architecture*** or the ***Fetch-Decode-Execute Architecture***.

- A Von Neumann architecture simply means that programs (together with data) are stored in main memory during execution.

- Not all computers follow the Von Neumann architecture. Some examples are ***processor array architecture***, ***multiprocessor architecture***, ***dataflow architecture***, and ***neural network architecture***.
- On the other hand, *Computer Organization* is the underlying implementation of the architecture which is transparent to the programmer. An architecture can have a number of organizational implementations:
  - Control Signals
  - Technologies
  - Device Implementations
- Most computers follow the *Von Neumann Architecture*. It is also known as the *Stored Program Architecture* or the *Fetch-Decode- Execute Architecture*.

- A computer follows the Von Neumann Architecture if it meets the following criteria:
  1. It has three basic hardware subsystems: a CPU, a main memory system, and an I/O system.
  2. It is a stored–program computer. Programs (together with data) are stored in main memory during execution.
  3. It carries out instructions sequentially.
  4. It has, or at least appears to have, a single path between the main memory and the control unit of the CPU.

ASSEMBLY LANGUAGE

- Types of Programming Language
  1. ***Machine Language***. The natural or primitive language that the computer actually understands. This programming language consists of 0's and 1's which makes programming very difficult.

Sample machine language program to add 5 and 3 (using the Intel microprocessor instruction set):

```
10111000
00000101
00000000
10111011
00000011
00000000
00010001
11011000
```

2. Assembly Language

A programming language that uses “abbreviations” or ***mnemonics*** in place of binary patterns in order to make the task of programming easier. Mnemonics are designed to be easy to remember and are a significant improvement over binary digits.

An assembly language program has to be converted to machine language before a computer can execute it. An ***assembler*** is a special program that translates assembly language mnemonics into machine language.

Sample assembly language program to add 5 and 3 (using the Intel microprocessor instruction set):

```
MOV AX, 05
MOV BX, 03
ADC AX, BX
```

NOTE: Both machine and assembly languages are ***low-level programming languages***.

3. High-Level Language

- A programming language that uses English-like commands or instructions. High-level languages are the easiest to use and contains many complicated or advanced instructions.
- A high-level language has to be converted to machine language before a computer can execute it. A ***compiler*** is a special program that translates high-level language instructions into machine language.

Examples of High-level Languages:

1. FORTRAN (FORmula TRANslation)
2. COBOL (COmmon Business-Oriented Language)
3. BASIC (Beginner's All-purpose Symbolic Instruction Code)
4. Pascal

Sample BASIC program to add 5 and 3:

```
LET A = 5
LET B = 3
LET C = A + B
```

- Advantages of using high-level languages over low-level languages:
  1. ***Easy to Learn***. Low-level languages are more cryptic than high-level language.
  2. ***Predefined Functions***. Most high-level languages provide many pre-defined functions and subroutines, thereby simplifying programming tasks.
  3. ***Portability***. Low-level languages are specific towards a certain processor. The instruction set of the Intel processors (IBM PC's and compatibles) is very much different from the instruction set of the Motorola processors (Apple Macintosh).
- Advantages of using low-level languages over high-level languages:
  1. ***Compact Code***. Programs are executed in their machine language format. Programs written in a high-level language should still be compiled and translated to machine language. Most compilers are not optimized to generate compact code.
  2. ***Speed***. This is directly related to compact code. The shorter the code, the shorter the execution time of the program.
  3. ***Flexible***. Low-level language does not constrain the programmer to follow a certain programming convention (i.e. modularity) nor a rigid coding constraint (i.e. the called routine should be placed before the calling routine).
- Application programs are now more and more complex and test the high-level language to its limit. High-level programmers now use assembly language-based subroutines to augment the capabilities of high-level languages.

BASIC OPERATIONAL CONCEPTS

- Example of a typical assembly language instruction:

```
ADD R0, LOCA
```

Add the operand at memory location *LOCA* to the operand in a register in the processor, *R0* and place the result into register *R0*.

The given instruction requires the performance of several steps:

1. The instruction must be transferred or fetched from the MM into the CPU.
2. The operand at *LOCA* must be fetched and added to the contents of *R0*.
3. The resultant sum is stored in register *R0*.

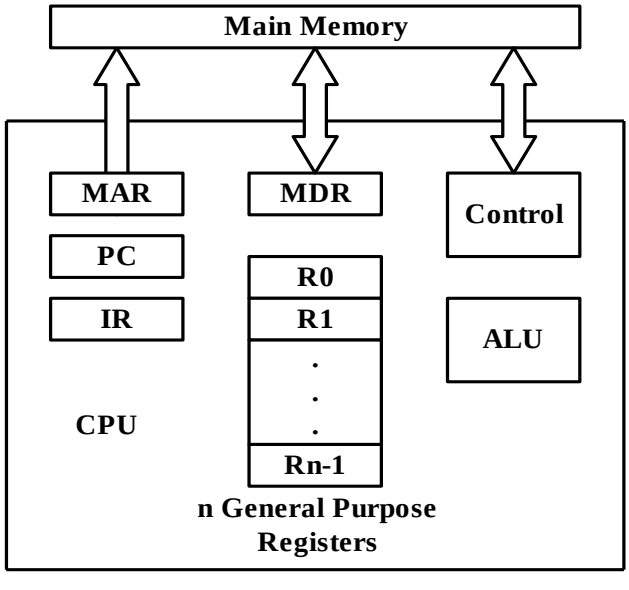
☞ In order to fetch/read an instruction or data from main memory:

1. The CPU first sends the address of the memory location to be read.
2. The CPU then issues or sends the *read* signal to the memory.
3. The word is then read out of memory and is loaded into a CPU internal register.

☞ In order to store/write data into main memory:

1. The CPU first sends the address of the memory location to be written.
2. The CPU then sends the *write* signal together with the data or word to be written to memory.

☞ Connections between the processor and the main memory:



The PC (***Program Counter***) contains the memory address of the instruction to be executed. During execution, the contents of the PC are updated to point to the next instruction.

The MAR (***Memory Address Register***) holds the address of the location to or from which data are to be transferred.

The MDR (***Memory Data Register***) contains the data to be written or read out of the addressed location.

The IR (***Instruction Register***) contains the instruction that is being executed.

☞ Operating Steps:

1. PC is set to point to the first instruction of the program (the operating system loads the memory address of the first instruction).
2. The contents of the PC are transferred to the MAR (which are automatically transmitted to the MM) and a *Read* signal is sent to the MM.
3. The addressed word is read out of MM and loaded into the MDR.
4. The contents of MDR are transferred to the IR. The instruction is ready to be decoded and executed.
5. During execution, the contents of the PC are incremented or updated to point to the next instruction.

   If operand or data needed by the instruction resides in MM:

   1. It will have to be fetched by sending its address to the MAR and initiating a read cycle.
   2. When the operand has been read from MM into the MDR, it may be transferred from the MDR to the ALU

      If result is to be stored in MM:

   1. The result is sent to the MDR.
   2. The address of the location where the result is to be stored is sent to the MAR and a write cycle is initiated.

BUS STRUCTURES

☞ A ***bus*** is a collection of wires that connect several devices within a computer system.

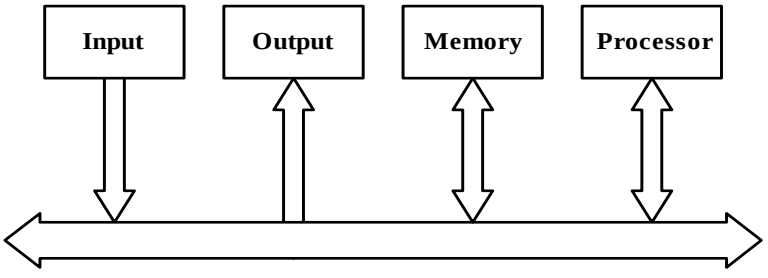
☞ When a word of data is transferred between units, all its bits are transferred in parallel.

☞ A computer must have some lines for addressing and control purposes.

☞ Three main groupings of lines:

1. ***Data Bus***. This is for the transmission of data.
2. ***Address Bus***. This specifies the location of data in MM.
3. ***Control Bus***. This indicates the direction of data transfer and coordinates the timing of events during the transfer.

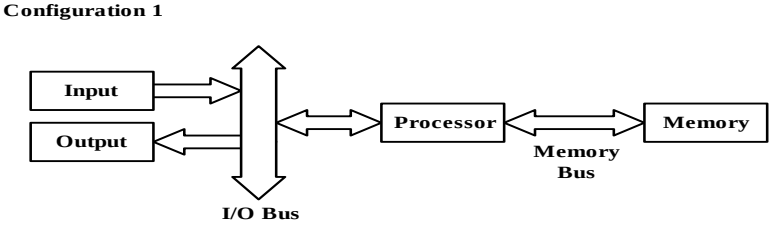
☞ Single-Bus Structure

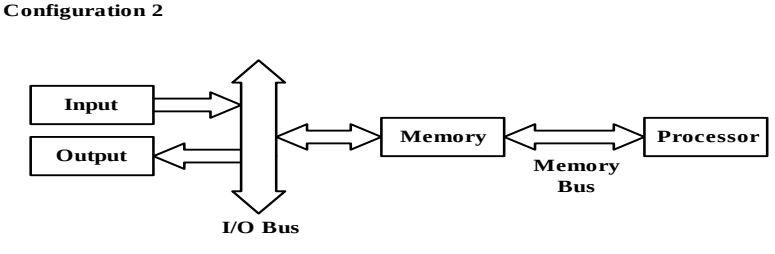


- All units are connected to a single bus, so it provides the sole means of interconnection.
- Only two units can actively use the bus at any given time.
- Bus control lines are used to arbitrate multiple requests for the use of the bus.

Buffer Registers are used to hold information during transfers. These prevent a high-speed processor from being locked to a slow I/O device during a sequence of data transfers.

☞ Two-Bus Structure





In the second configuration, I/O transfers are made directly to or from the memory. A special purpose processor called ***peripheral processor*** or ***I/O channel*** is needed as part of the I/O equipment to control and facilitate such transfers.

Watch:

Computer Organization and Architecture Lesson 1 – Introduction

COMPUTER ORGANIZATION AND ARCHITECTURE new

https://www.youtube.com/watch?v=t6_yhVTDfUE

https://www.youtube.com/watch?v=T9sbLbYHoOA

Read:

https://cnx.org/contents/K6UTbrNN@1/Introduction-to-Computer-Organization-and-Architecture

https://www.elprocus.com/difference-between-risc-and-cisc-architecture/

Activities/Assessments:

1. Have a recap on what have been discussed in this chapter.
2. Differentiate Computer Architecture from Computer Organization and its Structures.
3. The following are the key points:
   a. PC components
   b. PC hardware types
   c. Basic Structure of Computer Hardware and Software
   d. Addressing Modes
   e. Bus Interconnection

REFERENCES:

M. Morris Mano: Computer System Architecture, 2015

William Stalling: Computer organization and architecture, 2016

John P. Hayes: Computer Architecture and Organization, 2017

https://coelms.com

CHAPTER 2 - CENTRAL PROCESSING UNIT

Course Overview:

This course includes the study of CPU Structure and Function, Arithmetic and Logic Unit, Instruction Formats, Addressing Modes, Data Transfer and Manipulation, RISC and CISC and 64 Bit Processors.

Learning Objectives:

After successful completion of this lesson, you should be able to:

- Discuss the CPU Structure and its functions
- Learn the operation performed by the Arithmetic and Logic Unit
- Understand the different Instruction formats
- Discuss the Addressing modes of a processor
- Discuss the data transfer and manipulation

Course Materials:

- Computer
- Files (Central processing Unit)
- Demonstration Materials
    - Handouts
    - PowerPoint Presentation
    - Other Instructional materials

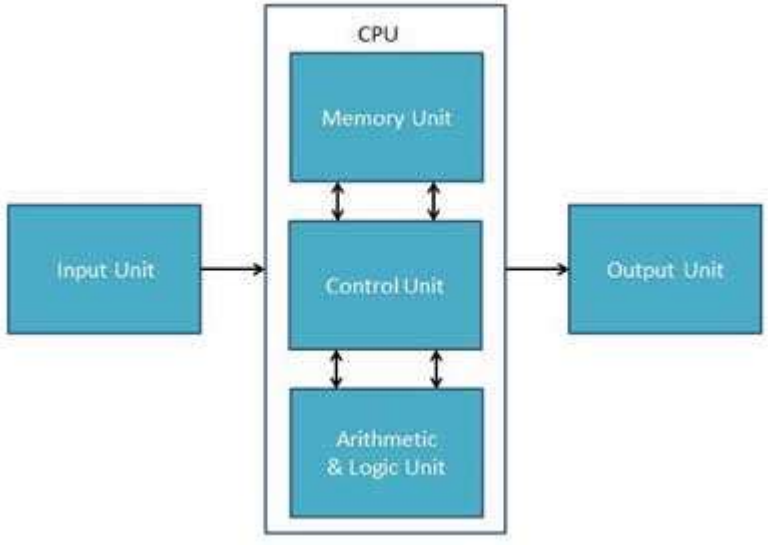
CENTRAL PROCESSING UNIT

Central Processing Unit (CPU) consists of the following features −

- CPU is considered as the brain of the computer.
- CPU performs all types of data processing operations.
- It stores data, intermediate results, and instructions (program).
- It controls the operation of all parts of the computer.

CPU itself has following three components.

- Memory or Storage Unit
- Control Unit
- ALU(Arithmetic Logic Unit)



Memory or Storage Unit

This unit can store instructions, data, and intermediate results. This unit supplies information to other units of the computer when needed. It is also known as internal storage unit or the main memory or the primary storage or Random Access Memory (RAM).

Its size affects speed, power, and capability. Primary memory and secondary memory are two types of memories in the computer. Functions of the memory unit are −

- It stores all the data and the instructions required for processing.
- It stores intermediate results of processing.
- It stores the final results of processing before these results are released to an output device.
- All inputs and outputs are transmitted through the main memory.

Control Unit

This unit controls the operations of all parts of the computer but does not carry out any actual data processing operations.

Functions of this unit are −

- It is responsible for controlling the transfer of data and instructions among other units of a computer.
- It manages and coordinates all the units of the computer.
- It obtains the instructions from the memory, interprets them, and directs the operation of the computer.
- It communicates with Input/Output devices for transfer of data or results from storage.

- It does not process or store data.

ALU (Arithmetic Logic Unit)

This unit consists of two subsections namely,

- Arithmetic Section
- Logic Section

Arithmetic Section

Function of arithmetic section is to perform arithmetic operations like addition, subtraction, multiplication, and division. All complex operations are done by making repetitive use of the above operations.

Logic Section

Function of logic section is to perform logic operations such as comparing, selecting, matching, and merging of data.

Computer Organization | Instruction Formats (Zero, One, Two and Three Address Instruction)

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

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

- Mode field which specifies how operand is to be founded.

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

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

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

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

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

1. Zero Address Instructions –

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

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

Postfixed : X = AB+CD+*

TOP means top of stack

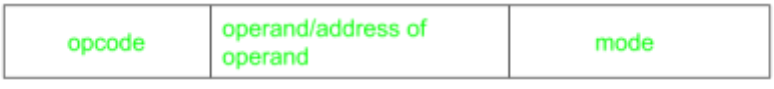
M[X] is any memory location

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D

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

2. **One Address Instructions –**

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



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

AC is accumulator

M[] is any memory location

M[T] is temporary location

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC
LOAD	C	AC = M[C]
ADD	D	AC = AC + M[D]
MUL	T	AC = AC * M[T]
STORE	X	M[X] = AC

3. **Two Address Instructions –**

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

Here destination address can also contain operand.

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

R1, R2 are registers

M[] is any memory location

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

- **Three Address Instructions –**

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

opcode	Destination address	Source address	Source address	mode

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

R1, R2 are registers

M[] is any memory location

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

Microprocessor - 8086 Addressing Modes

The different ways in which a source operand is denoted in an instruction is known as **addressing modes**. There are 8 different addressing modes in 8086 programming −

1. **Immediate addressing mode**

   The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.

Example

MOV CX, 4929 H, ADD AX, 2387 H, MOV AL, FFH

2. **Register addressing mode**

   It means that the register is the source of an operand for an instruction.

Example

```
MOV CX, AX   ; copies the contents of the 16-bit AX register into
             ; the 16-bit CX register),
ADD BX, AX
```

3. **Direct addressing mode**

The addressing mode in which the effective address of the memory location is written directly in the instruction.

Example

MOV AX, [1592H], MOV AL, [0300H]

4. Register indirect addressing mode

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.

Example

```
MOV AX, [BX]  ; Suppose the register BX contains 4895H, then the contents
              ; 4895H are moved to AX
ADD CX, {BX}
```

5. Based addressing mode

In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.

Example

MOV DX, [BX+04], ADD CL, [BX+08]

6. Indexed addressing mode

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.

Example

MOV BX, [SI+16], ADD AL, [DI+16]

7. Based-index addressing mode

In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register.

Example

ADD CX, [AX+SI], MOV AX, [AX+DI]

8. Based indexed with displacement mode

In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

Example

MOV AX, [BX+DI+08], ADD CX, [BX+SI+16]

DATA TRANSFER AND MANIPULATION

BASIC 8086/8088 INSTRUCTIONS

INTRODUCTION

- The instruction set of a microcomputer defines the basic operations that a programmer can make the device perform.
- There are 117 basic instructions for the 8086/88 chip.
- Classification of Instructions:
    1. Data Transfer Instructions
    2. Arithmetic Instructions
    3. Logic Instructions
    4. Shift Instructions
    5. Rotate Instructions
    6. Flag Control Instructions
    7. Jump Instructions
    8. String Instructions

DATA TRANSFER INSTRUCTIONS

- Data transfer instructions facilitate the movement of data either between registers or between a register and main memory. They do not affect the flags.

- **The MOV Instruction**

Format: MOV D, S

Action: D ⟵ [S]

Destination	Source	Example
Register	Register	MOV CX, BX
Register	MM	MOV CX, [BP+SI]
MM	register	MOV [BX], DX
Register	immediate	MOV CX, 80FEH
MM	immediate	MOV word ptr [BX], 1834H
seg reg	register	MOV DS, BX
Register	seg reg	MOV AX, CS
seg reg	MM	MOV SS, [1AFFH]
MM	seg reg	MOV [BP+SI+1000H], DS

Pointers:

1. The MOV instruction does not allow both source and destination operands to be memory locations (no memory to memory transfers). The instruction **MOV [BX], FILE** is invalid.
2. Both source and destination operands should be of the same data size. Therefore, the instruction **MOV AH, BX** is invalid.
3. Immediate data cannot be moved to a segment register. **MOV DS, 1800H** is therefore invalid. Instead, use:

   MOV AX, 1800H
   MOV DS, AX
4. The destination operand cannot be immediate data. **MOV 1800H, AX** is therefore invalid.
5. More often than not, register CS cannot be the destination operand.
6. If the destination operand is a memory location and the source operand is immediate data, the prefix ***byte ptr*** or ***word ptr*** should appear after the MOV instruction to denote the data size of the immediate operand.

☞ **The XCHG Instruction**

The XCHG (***Exchange***) instruction swaps the contents of the source and destination operands.

Format: XCHG D, S

Action: [D] ⟷ [S]

Destination	Source	Example
MM	register	XCHG [BX], CX
Register	register	XCHG AH, BL

Pointers:

1. The XCHG instruction does not allow both source and destination operands to be memory locations (no memory to memory transfers). The instruction **XCHG [BX], FILE** is therefore invalid.
2. Both source and destination operands should be of the same data size. Therefore, the instruction **XCHG AH, BX** is invalid.
3. Immediate addressing cannot be used with this instruction. The instruction **XCHG AX, 1800H** is invalid.
4. A data exchange can also be done using several MOV instructions:

```
MOV  CX, AX
MOV  DX, BX
MOV  AX, DX
MOV  BX, CX
```

However, four registers are needed in order to implement an exchange between two registers.

5. Segment registers cannot be used as operands in the XCHG instruction. **XCHG ES, BX** is therefore invalid. Instead use:

```
MOV CX, ES
XCHG BX, CX
MOV ES, CX
```

☞ **The PUSH Instruction**

The PUSH instruction decrements SP by 2 and then moves a word data from the source operand to the top of the stack pointed to by SP.

Format: PUSH S

Action: SP ⟵ SP - 2

[SP+ 1] ⟵ [SH]

[SP] ⟵ [SL]

Source	Example
register	PUSH AX
seg reg	PUSH DS
MM	PUSH BETA

Pointers:

1. The PUSH instruction pushes word-sized data only.

☞ **The POP Instruction**

POP transfers a word data pointed to by SP to the destination operand. Afterwards, the value of SP is incremented by 2.

Format: POP D

Action: DL ⟵ [SP]
DH ⟵ [SP + 1]
SP ⟵ SP + 2

Destination	Example
Register	POP AX
seg reg	POP DS
MM	POP BETA

Pointers:

1. The POP instruction pops word-sized data only.
2. **POP CS** is not allowed. However, the instruction **PUSH CS** is valid.

☞ **The LEA Instruction**

The LEA (***Load Effective Address***) instruction is for transferring the effective address (not data) of the source operand to the destination operand.

Format: LEA D, S

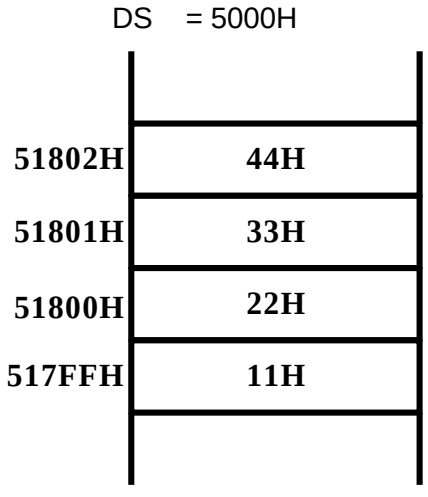
Action: D ⟵ EA of S

Pointers:

1. The source operand should always be one of the memory addressing modes, while the destination operand should always be a 16-bit register.
2. In the LEA instruction, the effective address is moved to the destination operand. Therefore, the destination operand is like a pointer.
3. Usually, registers BX, BP, DI, or SI are used as the destination operand since these registers could be used as offset registers.

Examples:

Assume the following:



4. If the **MOV DI, LIST** instruction is executed, the contents of memory location 51800H and 51801H are transferred to register DI. DI will therefore contain the word 3322H.
5. However, if the **LEA DI, LIST** instruction is executed, this will transfer the effective address of the source operand to register DI. Therefore, DI will now contain 1800H.

RISC and CISC

What is the Difference between RISC and CISC ARCHITECTURE?

The architecture of the Central Processing Unit (CPU) operates the capacity to function from "Instruction Set Architecture" to where it was designed. The architectural design of the CPU is Reduced instruction set computing (RISC) and Complex instruction set computing (CISC). A processor like CISC has the capacity to perform multi-step operations or addressing modes within one instruction set. It is the CPU design where one instruction works several low-level acts. For instance, memory storage, loading from memory, and an arithmetic operation. Reduced instruction set computing is a Central Processing Unit design strategy based on the vision that a basic instruction set gives great performance when combined with a microprocessor architecture that has the capacity to perform the instructions by using some microprocessor cycles per instruction. This article discusses the difference between RISC and CISC architecture. The hardware part of the Intel is named as Complex Instruction Set Computer (CISC), and Apple hardware is Reduced Instruction Set Computer (RISC).

Difference between RISC and CISC Architecture

Before we discuss the differences between the RISC and CISC architecture let us know about the concepts of RISC and CISC

RISC and CISC Processors

What is RISC?

A reduced instruction set computer is a computer that only uses simple commands that can be divided into several instructions that achieve low-level operation within a single CLK cycle, as its name proposes “Reduced Instruction Set”.

The RISC is a Reduced Instruction Set Computer microprocessor and its architecture includes a set of instructions that are highly customized. The main function of this is to reduce the time of instruction execution by limiting as well as optimizing the number of commands. So each command cycle uses a single clock cycle where every clock cycle includes three parameters namely fetch, decode & execute.

The kind of processor is mainly used to execute several difficult commands by merging them into simpler ones. RISC processor needs a number of transistors to design and it reduces the instruction time for execution. The best examples of RISC processors include PowerPC, SUN’s SPARC, RISC-V, Microchip PIC processors, etc.

RISC Architecture

The term RISC stands for ‘’Reduced Instruction Set Computer’’. It is a CPU design plan based on simple orders and acts fast.

This is a small or reduced set of instructions. Here, every instruction is expected to attain very small jobs. In this machine, the instruction sets are modest and simple, which help in comprising more complex commands. Each instruction is of a similar length; these are wound together to get compound

tasks done in a single operation. Most commands are completed in one machine cycle. This pipelining is a crucial technique used to speed up RISC machines.

Characteristics

The characteristics of RISC include the following.

- Pipeline architecture
- The number of instructions is restricted as well as decrease
- The instructions like load as well as store have right of entry to memory
- Addressing modes are less
- Instruction is uniform and its format can be simplified

Advantages

The advantages of the RISC processor include the following.

- The performance of this processor is good because of the easy & limited no. of the instruction set.
- This processor uses several transistors in the design so that making is cheaper.
- RISC processor allows the instruction to utilize open space on a microprocessor due to its simplicity.
- It is very simple as compared with another processor due to this; it can finish its task within a single clock cycle.

Disadvantages

The disadvantages of a CISC processor include the following.

- The performance of this processor may change based on the executed code because the next commands may depend on the earlier instruction for their implementation within a cycle.
- The complex instruction is frequently used by the compilers and programmers
- These processors need very quick memory to keep different instructions that use a huge collection of cache memory to react to the command within less time.

What is CISC?

It was developed by the Intel Corporation and it is Complex Instruction Set Computer. This processor includes a huge collection of simple to complex instructions. These instructions are specified in the level of assembly language level and the execution of these instructions takes more time.

A complex instruction set computer is a computer where single instructions can perform numerous low-level operations like a load from memory, an arithmetic operation, and a memory store or are accomplished by multi-step processes or addressing modes in single instructions, as its name proposes

"Complex Instruction Set ". So, this processor moves to decrease the number of instructions on every program & ignore the number of cycles for each instruction. It highlights to assemble complex instructions openly within the hardware as the hardware is always as compared with software. However, CISC chips are relatively slower as compared to RISC chips but utilize small instruction as compare with RISC. The best examples of the CISC processor include AMD, VAX, System/360 & Intel x86.

CISC Architecture

The term CISC stands for ''Complex Instruction Set Computer''. It is a CPU design plan based on single commands, which are skilled in executing multi-step operations. CISC computers have small programs. It has a huge number of compound instructions, which takes a long time to perform. Here, a single set of instructions is protected in several steps; each instruction set has additional than 300 separate instructions. Maximum instructions are finished in two to ten machine cycles. In CISC, instruction pipelining is not easily implemented.

Characteristics

The main characteristics of the RISC processor include the following.

- CISC may take more time to execute the code as compared with an only clock cycle.
- CISC supports high-level languages for simple compilation and complex data structure.
- It is collected with more addressing nodes, fewer registers normally from 5 to 20.
- For writing an application, less instruction is required
- The code length is very short, so it needs extremely small RAM.
- It highlights the instruction on hardware while designing as it is faster to design than the software.
- Instructions are larger as compared with a single word.
- It gives simple programming within assembly language.

Advantages

The **advantages of CISC** include the following.

- This processor will create a procedure to handle the usage of power that regulates the speed of clock & voltage.
- In the CISC processor, the compiler needs a small effort to change the program or statement from high-level to assembly otherwise machine language.
- A single instruction can be executed by using different low-level tasks
- It doesn't use much memory due to a short length of code.

- CISC utilizes less instruction set to execute the same instruction as the RISC.
- The instruction can be stored within RAM on every CISC

Disadvantages

The disadvantages of CISC include the following.

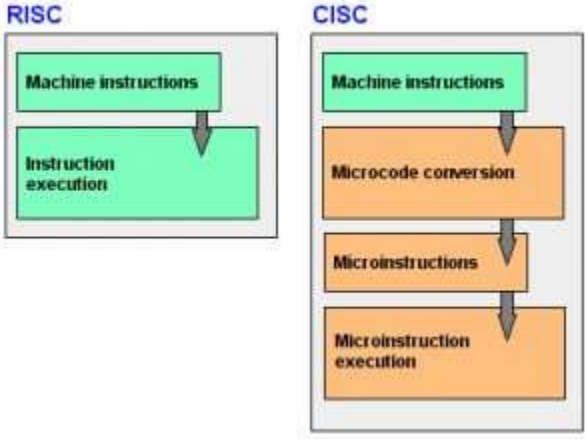
- The existing instructions used by the CISC are 20% within a program event.
- As compared with the RISC processor, CISC processors are very slow while executing every instruction cycle on every program.
- This processor use number of transistors as compared with RISC.
- The pipeline execution within the CISC will make it difficult to use.

Difference between RISC and CISC Architecture

RISC	CISC
1. RISC stands for Reduced Instruction Set Computer.	1. CISC stands for Complex Instruction Set Computer.
2. RISC processors have simple instructions taking about one clock cycle. The average clock cycle per instruction (CPI) is 1.5	2. CSIC processor has complex instructions that take up multiple clocks for execution. The average clock cycle per instruction (CPI) is in the range of 2 and 15.
3. Performance is optimized with more focus on software	3. Performance is optimized with more focus on hardware.
4. It has no memory unit and uses separate hardware to implement instructions..	4. It has a memory unit to implement complex instructions.
5. It has a hard-wired unit of programming.	5. It has a microprogramming unit.
6. The instruction set is reduced i.e. it has only a few instructions in the instruction set. Many of these instructions are very primitive.	6. The instruction set has a variety of different instructions that can be used for complex operations.
7. The instruction set has a variety of different instructions that can be used for complex operations.	7. CISC has many different addressing modes and can thus be used to represent higher-level programming language statements more efficiently.
8. Complex addressing modes are synthesized using the software.	8. CISC already supports complex addressing modes
9. Multiple register sets are present	9. Only has a single register set
10. RISC processors are highly pipelined	10. They are normally not pipelined or less pipelined
11. The complexity of RISC lies with the compiler that executes the program	11. The complexity lies in the microprogram
12. Execution time is very less	12. Execution time is very high
13. Code expansion can be a problem	13. Code expansion is not a problem

14. The decoding of instructions is simple.	14. Decoding of instructions is complex
15. It does not require external memory for calculations	15. It requires external memory for calculations
16. The most common RISC microprocessors are Alpha, ARC, ARM, AVR, MIPS, PA-RISC, PIC, Power Architecture, and SPARC.	16. Examples of CISC processors are the System/360, VAX, PDP-11, Motorola 68000 family, AMD, and Intel x86 CPUs.
17. RISC architecture is used in high-end applications such as video processing, telecommunications, and image processing.	17. CISC architecture is used in low-end applications such as security systems, home automation, etc.

- The machine performance reduces because of the low speed of the clock.



Key Differences between RISC and CISC

The key differences between RISC and CISC include the following.

- The size of an instruction set is small as compared with RISC.
- In RISC, the CPU control can be done with hardwired without comprising a control memory whereas CISC is micro coded that uses ROM, however, the current CISC processor also utilizes hardwired control.
- RISC processor works with 32-bits for each instruction and frequently based on the register while CISC utilizes an uneven format that ranges from 16 bits to 64 bits for each instruction.
- RISC architecture includes the design of instruction cache and split data whereas CISC architecture includes a unified cache intended for data & instructions, even though most recent designs also utilize split caches.
- In the RISC processor, the mechanism of memory used is register to register including the instructions like STORE & independent LOAD. In CISC, the mechanism of memory used is memory to memory to execute different operations including the instructions like LOAD & STORE.

- The general purpose registers used in the RISC processor are 32 to 192 whereas RISC uses 8 to 24 GPR's.
- In the RISC processor, the single clock is used, and addressing modes are limited whereas, in CISC, it uses the multi clock, and addressing modes ranges from 12 to 24.
- The **difference between RISC and CISC instruction set** is, RISC ISA highlights software as compared with hardware. The instruction set of the RISC processor uses more efficient software like code or compilers through fewer instructions. CISC ISAs employ a number of transistors within the hardware to execute several instructions as well as additional complex instructions also.

The **advantages of RISC over CISC** include the following.

In the current developments of computer processors, the RISC (reduced instruction set computer) microprocessor is the most frequently used and significant one. Beneath certain conditions, the devices based on this processor will offer important benefits over CISC (complex instruction set computer). In the above, a brief comparison between both the processors is discussed.

The RISC processor performance is two to four times higher as compared with CISC processors due to the basic instruction set. The architecture of this processor uses very little space because of the decreased instruction set and this will make additional functions such as memory management or floating-point arithmetic units on a similar chip.

This article discusses the concepts of RISC, CISC, and differences. When the first microprocessors, as well as microcontrollers, were introduced, there is no better and suitable architecture. Once these processors were implemented, the CISC architecture is used mostly due to the lack of software support in the RISC processor. This is mainly doing to build all their hardware as well as software back well-suited through their first 8086 processors. We hope that you have got a better understanding of this concept. Furthermore, for any doubts regarding this concept, or implementation of any electrical and electronic projects, please give your feedback by commenting on the comment section below.

64-bit processor

The 64-bit computer originated in 1961 when IBM created the IBM 7030 Stretch supercomputer. However, it was not put into use in home computers until the early 2000s. Microsoft released a 64-bit version of Windows XP to be used on computers with a 64-bit processor. Windows Vista, Windows 7, and Windows 8 also come in 64-bit versions.

Note

A computer with a 64-bit processor can have a 64-bit or 32-bit version of an operating system installed. However, with a 32-bit operating system, the 64-bit processor would not run at its full capability.

Note

On a computer with a 64-bit processor, you cannot run a 16-bit legacy program. Many 32-bit programs works with a 64-bit processor and operating system, but some older 32-bit programs may not function properly, or at all, due to limited or no compatibility.

Differences between a 32-bit and 64-bit CPU

A big difference between 32-bit processors and 64-bit processors is the number of calculations per second they can perform, which affects the speed at which they can complete tasks. 64-bit processors come in dual-core, quad-core, six-core, and eight-core versions for home computing. Multiple cores allow for an increased number of calculations per second that can be performed, which increases the processing power and helps make a computer run faster. Software programs that require many calculations to function smoothly can operate faster and more efficiently on the multi-core 64-bit processors, for the most part.

Another big difference between 32-bit processors and 64-bit processors is the maximum amount of memory (RAM) that is supported. 32-bit computers support a maximum of 4 GB (232 bytes) of memory, whereas 64-bit CPUs can address a theoretical maximum of 18 EB (264 bytes). However, the practical limit of 64-bit CPUs (as of 2018) is 8 TB of addressable RAM.

High amounts of RAM are especially useful for software used in graphic design, engineering, and video editing as these programs have to perform many calculations to render their images.

One thing to note is that 3D graphics programs and games do not benefit much, if at all, unless the program is 64-bit. A 32-bit processor is adequate for any program written for a 32-bit processor. In the case of computer games, if it's designed for 32-bit processors, you'll get a lot more performance by upgrading the video card instead of getting a 64-bit processor. However, if the game is designed for 64-bit processors, upgrading to a 64-bit processor makes a big improvement in the performance of the game.

In the end, 64-bit processors are commonplace in home computers today. Most manufacturers build computers with 64-bit processors due to cheaper prices and because more users are now using 64-bit operating systems and programs. Computer parts retailers are offering fewer and fewer 32-bit processors and soon may not offer any at all.

Watch:

https://www.youtube.com/watch?v=FZGugFqdr60

https://www.youtube.com/watch?v=_EKgwOAAWZA

Read:

https://cnx.org/contents/K6UTbrNN@1/Introduction-to-Computer-Organization-and-Architecture

Activities/Assessments:

1. Have a recap on what have been discussed in this chapter.
2. Give at least 1 example of each addressing modes.
3. Perform each Data Transfer instructions.
4. Differentiate CISC from RISC
5. The following are the key points:
   a. CPU Structure and Function
   b. Arithmetic and Logic Unit
   c. Instruction Formats
   d. Addressing Modes
   e. Data Transfer and Manipulation
   f. RISC and CISC
   g. 64 Bit Processors

Quiz No. 1

MULTIPLE CHOICE [20 points]. Choose the letter that corresponds to your answer. Write your answer on the space provided. Use CAPITAL letters only.

B 1. It is the first microprocessor released by the Intel in 1971.
A. 4040 B. 4004 C. 4400 D. None of the above

D 2. It is the number of memory locations in 8088/8086.
A. 1,048,574 bytes B. 524,255 16-bit C. 1,048,578 bytes D. None of the above

C 3. It is the part of 8088/8086 microprocessor that prefetches the succeeding instructions while executing the current operation.
A. Instruction pointer B. Execution unit C. Prefetch Queue D. None of the above

C 4. The physical memory of the 8088 contains two banks of memory. This statement is:
A. Always true. C. False, it contains one bank only.
B. False, it contains three banks of memory. D. False, it doesn't contain bank of memory.

D 5. It is the total number of memory locations dedicated for functions, storage or pointers to interrupt, and reserved for future use.
A. 128 bytes B. 108 bytes C. 16 bytes D. None of the above

D 6. It is a general purpose register used to hold temporary result of an arithmetic or logic operation.
A. Base register B. Data register C. Count register D. None of the above

A 7. It is the total number of unused bits in status register.
A. 7 bits B. 16 bits C. 9 bits D. None of the above

B 8. It is the segment used for string instructions.
A. Code segment B. Extra segment C. Stack segment D. Data segment

B 9. It is the initial value of the stack pointer upon start-up of the microprocessor.
A. 0000H B. FFFFH C. A000H D. None of the above

C 10. It is an addressing mode used when the source operand represents a constant data.
A. Register addressing B. Direct addressing C. Immediate addressing D. None of the above

A 11. It is an addressing modes in which the operand to be accessed is specified as residing in an internal register of the 8086/8088.
A. Register addressing B. Direct addressing C. Immediate addressing D. Register Indirect

B 12. It is an addressing mode in which the location following the instruction opcode holds an effective memory address (EA) instead of data.
A. Register addressing B. Direct addressing C. Immediate addressing D. Register Indirect

B 13. It is an entity in which an operand is performed.
A. Opcode B. Operand C. Register D. None of the above

A 14. It is a high-speed memory internal to the CPU.
A. Register B. Flip-flops C. System bus D. None of the above

B 15. It is a computer oriented language whose instructions is usually in one-to-one correspondence with computer instructions and provides facilities such as the use of microinstructions.
A. Machine language B. Assembly language C. Assembler D. None of the above

A 16. It is the name of the CPU register in a single-address instruction format.
A. Accumulator B. Base register C. Count register D. Data Register

C 17. It is the numeric value that is used as a reference in the calculation of addresses in the execution of the computer program
A. Base address B. Offset address C. Physical address D. None of the above

program conter D 18. It is a Special-purpose register used to holds the address of the next instruction to be executed.
A. Instruction address register C. Instruction register
B. General-purpose register D. None of the above

D 19. It is considered as the primary port on all computer system.
A. Mouse port B. Printer port C. Keyboard port D. None of the above

A 20. It is a segment register that contains the program or code in the next instruction executed by the 8086/8088 is generated by adding the contents of IP to the contents CS x 10H.
A. Code segment B. Data segment C. Stack segment D. Extra Segment

REFERENCES:

M. Morris Mano: Computer System Architecture, 2015

William Stalling: Computer organization and architecture, 2016

John P. Hayes: Computer Architecture and Organization, 2017

https://coelms.com

CHAPTER 3 - CONTROL UNIT

Course Overview:

This course includes the study of Control Memory, Address Sequencing, Computer Configuration, Microinstruction Format, Symbolic Microinstruction, Symbolic Micro Program, CU Operations, and Design of CU.

Learning Objectives:

After successful completion of this lesson, you should be able to:

- Discuss the function and operation of the Control Unit
- Identify the various components that interact in the control unit.
- Discuss the microinstruction format
- Discuss the designing of the control unit

Course Materials:

- Computer
- Files (Control Unit)
- Demonstration Materials
    - Handouts
    - PowerPoint Presentation
    - Video clips
    - Other Instructional materials

CONTROL MEMORY

Control memory is memory inside the CPU or other control unit. You cannot see it on the motherboard, or even by looking at the CPU or control chip. This memory holds microinstructions. If the memory can be written to, it is called the Writeable Control Store. This memory hold the steps or inner-workings of the CPU itself. These days, RISC based CPUs do not use microinstructions, because by hardwiring the instructions they can get faster execution.

Main memory is outside the CPU. You can see it plugged into the motherboard. Main memory is accessed through the MMU and the cache of the CPU. Access to control memory is only internal to the chip itself.

What is memory controller?

Memory controller is a logical block that performs reads/writes from a memory based on the memory technology. Most commonly memory controller refers to main memory (DRAM) controller.

The DRAM memory controller translates addresses coming from a CPU read/write into a DRAM address (bank,page, row, column bits) based on the type of memory attached , its size, organization etc. It also needs to follow and be compliant with the DRAM protocol and timings (DDR2/DDR3/4 protocol and various timing requirements between sub commands).

There will also be logic support to enhance performance of memory access (e.g - tracking open/closed pages, read/write ordering, write to read data forwarding, automatic vs adaptive page closing etc). It also needs to ensure the DRAMs are refreshed at regular intervals, data returned from DRAMs checked for errors (and correction if possible).

Another type of memories that are commonly used are Flash memory devices like USB sticks or SSD drives - In this case, there needs to be a Flash/SSD Memory controller that performs write/read/erase operations to the Flash memory (NAND or NOR). This is little more complicated and normally also is intervened by firmware - Flash memory does not support random access, are typically accessed in blocks/pages and the controllers also need to be taking care of reliability and performance

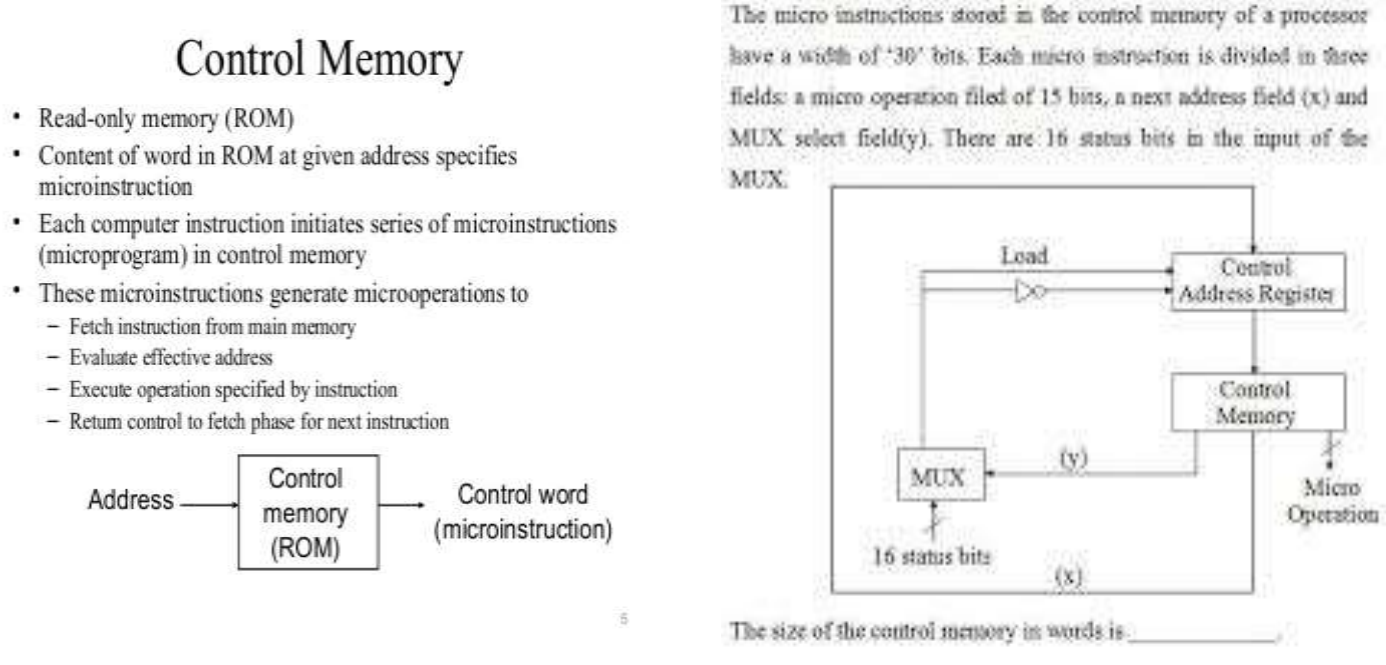
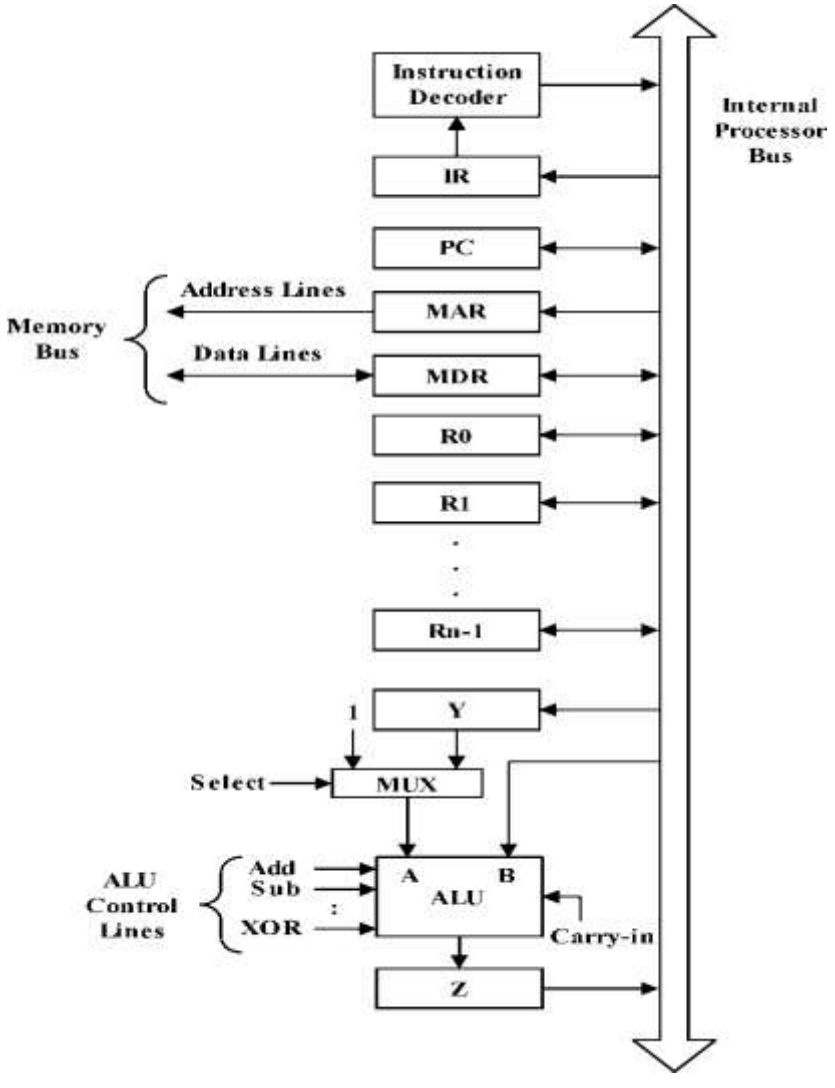


Image of a Control Memory

ADDRESS SEQUENCING

☞ Single-bus organization of the data paths inside the processing unit:



- In this organization, the arithmetic and logic unit (ALU) and all the registers are interconnected via a single common bus. This bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.
- The data and address lines of the external memory bus are connected to the internal processor via the MDR and the MAR respectively
- Two registers (Y and Z) have not been mentioned before. These registers are transparent to the programmer; that is, the programmer need not be concerned with them, because they are never referenced directly by any instruction. They are used only by the processor for temporary storage during execution of some instructions.

- The multiplexer (MUX) selects either the output of register Y or a constant 1 to be provided as input A of the ALU. The constant 1 is used to increment the contents of the program counter.
- The instruction decoder unit is responsible for implementing the actions specified by the instruction loaded in the IR register. The decoder generates the control signals needed to select the registers involved and direct the transfer of data.
- The registers, the ALU, and the interconnecting bus are collectively referred to as the data path.
- With a few exceptions, an instruction can be executed by performing one or more of the following operations in some specified sequence:
    1. Transfer a word of data from one processor register to another or to the ALU.
    2. Perform an arithmetic or logic operation and store the result in a processor register.
    3. Fetch the contents of a given memory location and load them into a processor register.
    4. Store a word of data from a processor register into a given memory location.

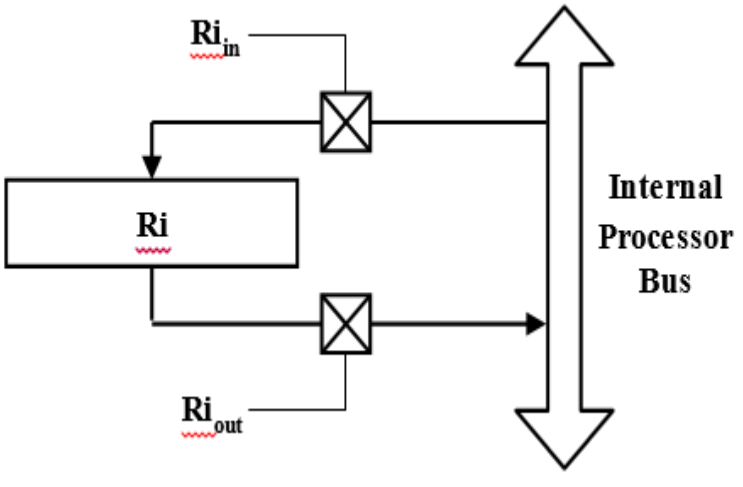
Register Transfers

To enable data transfers between various blocks in a common data bus, input and output gating must be provided. The input and output gates for register Ri are controlled by the signals Riin and Riout, respectively.

If Riin = 1, the data available on the common bus is loaded into Ri.

If *Riin* = 1, the data available on the common bus is loaded into *Ri*.

If *Riout* = 1, the contents of register *Ri* are placed on the bus.



Example:

Assume that the address of the memory location to be accessed is in R1 and the memory data is to be loaded into R2. The actions needed to execute this instruction are:

1. MAR ← [R1]
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory bus
5. R2 ← [MDR]

The actions may be carried out as separate steps, but some can be combined into a single step. Each action can be completed in one clock cycle, except action 3 which requires one or more clock cycles, depending on the speed of the addressed device.

Example:

Assume that the data word to be stored in memory is in R2 and the memory address is in R1.

1. R1out, MARin
2. R2out, MDRin, Write
3. MDRoutE, WMFC

As in the case of the read operation, the Write control signal causes the memory bus interface to issue a Write command on the memory bus. The processor remains in step 3 until the memory operation is completed and an MFC response is received.

EXECUTION OF A COMPLETE INSTRUCTION

- Assume the following instruction: **R1 ← [R1] + [[R2]]**
- The contents of register R1 are added to the contents of a memory location (the address of which is in register R2) and store the results in register R1.

Executing this instruction requires the following actions:

1. Fetch the instruction.
2. Fetch the first operand (the contents of the memory location pointed to by R2).
3. Perform the addition.
4. Load the result into R1.

Control Sequence for R1 ← [R1] + [[R2]]

1. PCout, MARin, Read, Select1, Add, Zin
2. Zout, PCin, Yin, WMFC
3. MDRout, IRin

4. $R2_{out}$, MAR_{in}, Read
5. $R1_{out}$, Y_{in}, WMFC
6. MDR_{out}, SelectY, Add, Z_{in}
7. Z_{out}, $R1_{in}$, End

- Step 1. The instruction fetch operation is initiated by loading the contents of the PC into the MAR and sending a Read request to the memory. The select signal is set to Select1, which causes the multiplexer MUX to select the constant 1. This value is added to the operand at input B, which is the contents of the PC, and the result is stored in register Z (the PC is incremented by 1).
- Step 2. The updated value of the PC (which is in register Z) is moved back to the PC. Note that step 2 begins immediately after the memory Read is requested, without waiting for the memory function to be completed. Step 3, however, has to be delayed until the MFC signal is received.
- Step 3. The word fetched from the memory is loaded into the IR. The instruction decoding circuit interprets the contents of the IR at the beginning of step 4. This enables the control circuitry to choose the appropriate signals for the remainder of the control sequence.

  - Steps 1 through 3 constitute the instruction fetch phase, which is the same for all instructions. Steps 4 through 7 constitutes the execution phase.
  - The contents of register R3 are transferred to the MAR in step 4, and a memory read is initiated.
  - Then the contents of R1 are transferred to register Y in step 5, to prepare for the addition operation.
  - When the Read operation is completed, the memory operand is available in register MDR, and the addition operation is performed in step 6. The contents of the MDR are gated to the bus, and thus also to the B input of the ALU, and register Y is selected as the second input to the ALU by choosing SelectY. The sum is stored in register Z.
  - The sum is then transferred to R1 in step 7. The End signal causes a new instruction fetch cycle to begin by returning to step 1.
  - Take note that in step 2, the updated contents of PC are also loaded to register Y. This has no purpose in the execution of the Add instruction. But in Branch instructions the updated value of PC is needed to compute the Branch target address. To speed up the execution of Branch instructions, this value is copied into register Y in step 2. Since step

2 is part of the fetch phase, the same action will be performed for all instructions. This does not cause any harm because register Y is not used for any other purpose at that time.

Watch:

https://www.youtube.com/watch?v=IzWvFfiQPbA

https://www.youtube.com/watch?v=ZyPewh04OzQ

Read:

https://cnx.org/contents/K6UTbrNN@1/Introduction-to-Computer-Organization-and-Architecture

https://www.elprocus.com/difference-between-risc-and-cisc-architecture/

Activities/Assessments:

1. Have a recap on what have been discussed in this chapter.
2. Watch the video and discuss briefly how the Control Unit of the computer works?
3. The following are the key points:
   a. Control Memory
   b. Address Sequencing
   c. Computer Configuration
   d. Microinstruction Format
   e. Symbolic Microinstruction
   f. Symbolic Micro Program
   g. CU Operations
   h. Design of CU

Exercise No. 1

Write the complete sequence of control steps required for each of the following notations/instructions.

1. **R2 ← [R2] + [[R1]]**
2. **R1 ← [R1] +[[[R2]]]**

Quiz No. 2

Choose the letter of the correct answer. Write your answer on the space provided. If the answer is not found among the choices, write **RM**. Use **CAPITAL LETTERS** only.

B 1. It is equal to the product of instruction count, cycle per instruction and the period.
A. Memory time B. CPU time C. Cache time D. MIPS

A 2. It is the time needed to complete one memory reference (read or write).
A. Memory cycle B. CPU cycle C. Register cycle D. Data cycle

B 3. It is often called the Instruction Set Processor or simply processor.
A. Memory Unit B. Processing Unit C. MAR D. Program Counter

RM 4. It contains the memory address of the instruction to be executed.
A. Memory Unit B. Processing Unit C. MAR D. Program Counter

C 5. It holds the address of the location to or from which data are to be transferred.
A. Memory Unit B. Processing Unit C. MAR D. Program Counter

A 6. It contains the instruction that is being executed.

C 7. A. Instruction Register B. Processing Unit C. MAR D. Program Counter
It contains the data to be written or read out of the addressed location.

A 8. A. Memory Unit B. Processing Unit C. MDR D. Program Counter
To execute a program, the processor fetches how many at a time and performs the functions specified?

B 9. A. One B. Two C. Three D. Four
The registers, the ALU and the interconnecting bus are collectively referred to as:

B 10. A. Data B. Data path C. MAR D. Bus
It is combinational circuit which functions as data selector.

B 11. A. Decoder B. MUX C. ALU D. Encoder
It can be enhanced with better hardware technology, innovative architectural features, and efficient resource management.

A 12. A. Program behavior B. Machine capability C. All of the above D. None of the above
It is affected by algorithm design, data structures, language efficiency, programmers skill, and compiler technology.

C 13. A. Program behavior B. Machine capability C. All of the above D. None of the above
It is a combinational circuit that performs arithmetic and logical operations.

A 14. A. Decoder B. MUX C. ALU D. Encoder
It is a sequence of data including input, partial, or temporary results, called for by the instruction stream.

B 15. A. Data Stream B. Instruction Stream C. Register Stream D. Step decoder
It is a sequence of instructions as executed by a machine.

A 16. A. Data Stream B. Instruction Stream C. Register Stream D. Step decoder
It is the most universally accepted method of classifying computers.

D 17. A. Flynn's Classification of Computers C. Von Neumann Architecture
B. Flynn's Computer Architecture D. Von Neumann Classification of Computers
It is a part of a computer where data are fetched.

C 18. A. Control Unit B. Instruction Unit C. Processor Unit D. Memory Unit
It is a part of a computer where instructions are being executed.

C 19. A. Control Unit B. Instruction Unit C. Processor Unit D. Memory Unit
Which register contains the data to be written or read out of the address location?

A 20. A. IR B. MAR C. MDR D. PC
In order to fetch data from memory, the first thing the processor does is:
A. send the memory address. C. Both (A) and (B)
B. send the memory read signal. . Either (A) or (B)

REFERENCES:

M. Morris Mano: Computer System Architecture, 2015

William Stalling: Computer organization and architecture, 2016

John P. Hayes: Computer Architecture and Organization, 2017

https://coelms.com

CHAPTER 4 - PIPELINE AND VECTOR PROCESSING

Course Overview:

This course includes the study of Pipelining, Parallel Processing, Arithmetic Pipeline, Instruction Pipeline, RISC Pipeline, Vector Processing and Array Processing

Learning Objectives:

After successful completion of this lesson, you should be able to:

- Discuss the Pipelining concept
- Discuss the Vector processing
- Learn the RISC Pipelining
- Learn the Array Processing

Course Materials:

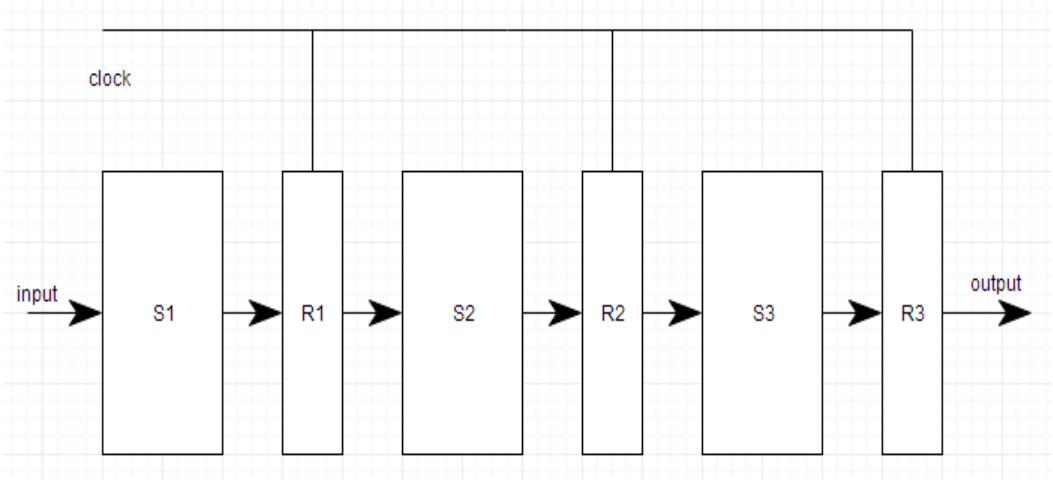
- Computer
- Files (PIPELINE AND VECTOR PROCESSING)
- Demonstration Materials
    - Handouts
    - PowerPoint Presentation
    - Video clips
    - Other Instructional materials

WHAT IS PIPELINING?

Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing.

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.

Pipelining increases the overall instruction throughput. In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.



Pipeline system is like the modern day assembly line setup in factories. For example in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm.

Types of Pipeline

It is divided into 2 categories:

1. Arithmetic Pipeline
2. Instruction Pipeline

1. Arithmetic Pipeline

Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc. For example: The input to the Floating Point Adder pipeline is:

```
X = A*2^a

Y = B*2^b
```

Here A and B are mantissas (significant digit of floating point numbers), while **a** and **b** are exponents.

The floating point addition and subtraction is done in 4 parts:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract mantissas

4. Produce the result.

Registers are used for storing the intermediate results between the above operations.

2. Instruction Pipeline

In this a stream of instructions can be executed by overlapping *fetch*, *decode* and *execute* phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system.

An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

Pipeline Conflicts

There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given below:

1. Timing Variations

All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.

2. Data Hazards

When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results.

3. Branching

In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

4. Interrupts

Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

5. Data Dependency

It arises when an instruction depends upon the result of a previous instruction but this result is not yet available.

Advantages of Pipelining

1. The cycle time of the processor is reduced.
2. It increases the throughput of the system
3. It makes the system reliable.

Disadvantages of Pipelining

1. The design of pipelined processor is complex and costly to manufacture.
2. The instruction latency is more.

VECTOR (ARRAY) PROCESSING AND SUPERSCALAR PROCESSORS

A **Scalar processor** is a normal processor, which works on simple instruction at a time, which operates on single data items. But in today's world, this technique will prove to be highly inefficient, as the overall processing of instructions will be very slow.

What is Vector (Array) Processing?

There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require vast number of computations on multiple data items that will take a conventional computer (with scalar processor) days or even weeks to complete.

Such complex instructions, which operates on multiple data at the same time, requires a better way of instruction execution, which was achieved by Vector processors.

Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like **ADD A to B, and store into C** are not practically efficient.

Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So until the data is found, the CPU would be sitting ideal, which is a big performance issue.

Hence, the concept of **Instruction Pipeline** comes into picture, in which the instruction passes through several sub-units in turn. These sub-units perform various independent functions, **for example**: the **first** one decodes the instruction, the **second** sub-unit fetches the data and the **third** sub-unit

performs the math itself. Therefore, while the data is fetched for one instruction, CPU does not sit idle, it rather works on decoding the next instruction set, ending up working like an assembly line.

Vector processor, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time.

A normal scalar processor instruction would be ADD A, B, which leads to addition of two operands, but what if we can instruct the processor to ADD a group of numbers(from 0 to n memory location) to another group of numbers (let's say, n to k memory location). This can be achieved by vector processors.

In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

Applications of Vector Processors

Computer with vector processing capabilities are in demand in specialized applications. The following are some areas where vector processing is used:

1. Petroleum exploration.
2. Medical diagnosis.
3. Data analysis.
4. Weather forecasting.
5. Aerodynamics and space flight simulations.
6. Image processing.
7. Artificial intelligence.

Superscalar Processors

It was first invented in 1987. It is a machine which is designed to improve the performance of the scalar processor. In most applications, most of the operations are on scalar quantities. Superscalar approach produces the high performance general purpose processors.

The main principle of superscalar approach is that it executes instructions independently in different pipelines. As we already know, that Instruction pipelining leads to parallel processing thereby speeding up the processing of instructions. In Superscalar processor, multiple such pipelines are introduced for different operations, which further improves parallel processing.

There are multiple functional units each of which is implemented as a pipeline. Each pipeline consists of multiple stages to handle multiple instructions at a time which support parallel execution of instructions.

It increases the throughput because the CPU can execute multiple instructions per clock cycle. Thus, superscalar processors are much faster than scalar processors.

A **scalar processor** works on one or two data items, while the **vector processor** works with multiple data items. A **superscalar processor** is a combination of both. Each instruction processes one data item, but there are multiple execution units within each CPU thus multiple instructions can be processing separate data items concurrently.

While a superscalar CPU is also pipelined, there are two different performance enhancement techniques. It is possible to have a non-pipelined superscalar CPU or pipelined non-superscalar CPU. The superscalar technique is associated with some characteristics, these are:

1. Instructions are issued from a sequential instruction stream.
2. CPU must dynamically check for data dependencies.
3. Should accept multiple instructions per clock cycle.

VECTOR (ARRAY) PROCESSOR AND ITS TYPES

Array processors are also known as multiprocessors or vector processors. They perform computations on large arrays of data. Thus, they are used to improve the performance of the computer.

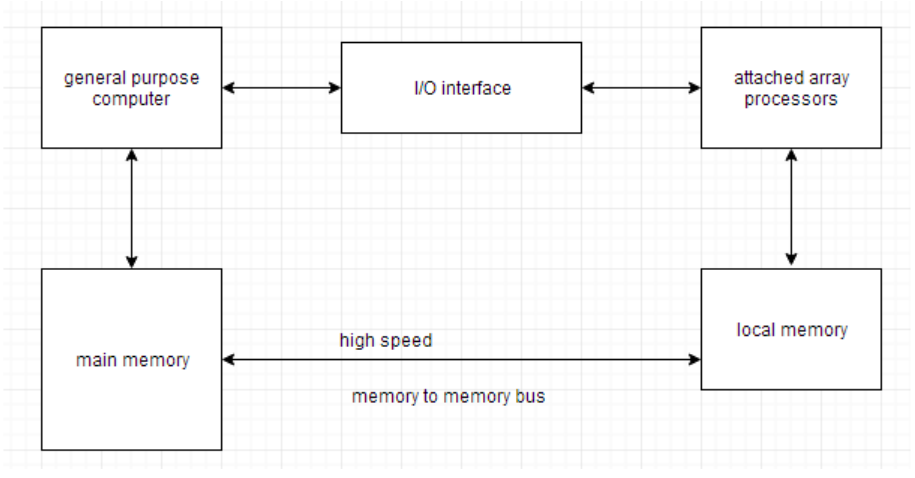
Types of Array Processors

There are basically two types of array processors:

1. Attached Array Processors
2. SIMD Array Processors

1. Attached Array Processors

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units.



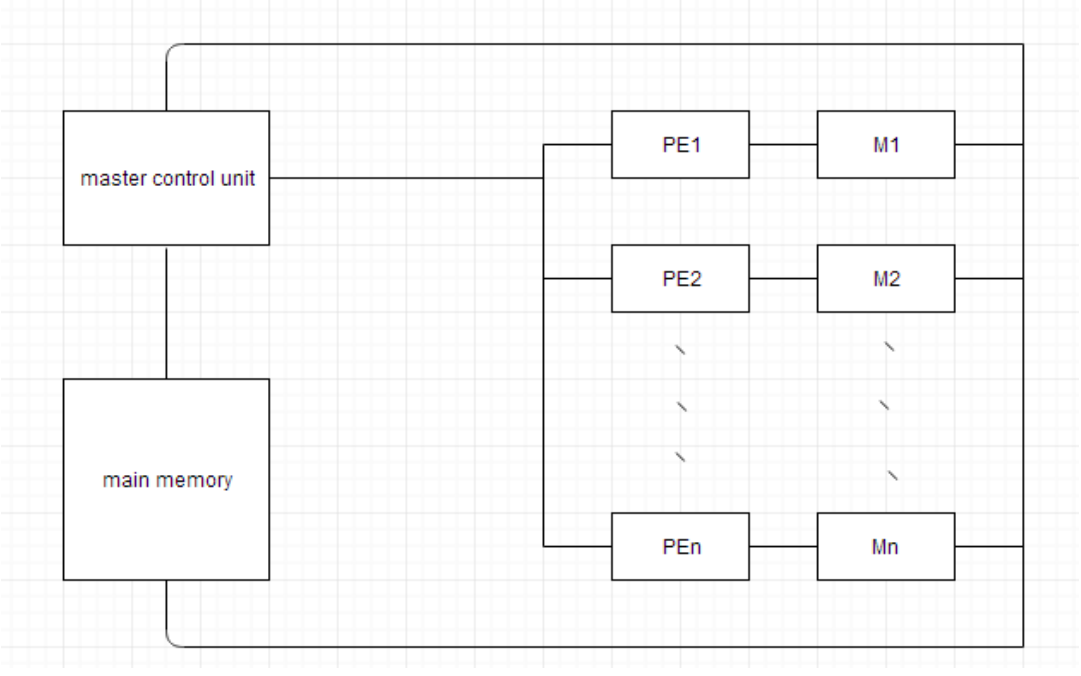
SIMD Array Processors

SIMD is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an **ALU** and **registers**. The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are send to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the **ILLIAC IV** computer developed by the **Burroughs corps**. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.



Why use the Array Processor

- Array processors increases the overall instruction processing speed.
- As most of the Array processors operates asynchronously from the host CPU, hence it improves the overall capacity of the system.
- Array Processors has its own local memory, hence providing extra memory for systems with low memory.

Watch:

https://www.youtube.com/watch?v=lTgVoICqKEs

Read:

https://www.studytonight.com/computer-architecture/input-output-organisation

https://www.studytonight.com/computer-architecture/array-processor

Activities/Assessments:

1. Have a recap on what have been discussed in this chapter.
2. Watch the video and discuss briefly how Pipeline and Vector Processing of Computer Architecture work.
3. Illustrate an application of the pipelining and vector processing.
4. The following are the key points:
   a. Pipelining

b. Parallel Processing
c. Arithmetic Pipeline
d. Instruction Pipeline
e. RISC Pipeline
f. Vector Processing
g. Array Processing

CHAPTER 5 - COMPUTER ARITHMETIC

Course Overview:

This course includes the study of Addition Algorithm, Subtraction Algorithm, Multiplication Algorithm and Division Algorithm and how these operation are being achieved or executed.

Learning Objectives:

After successful completion of this lesson, you should be able to:

- Understand the concept of computer arithmetic
- Discuss the addition algorithm
- Discuss the subtraction algorithm
- Discuss the multiplication algorithm
- Discuss the division algorithm

Course Materials:

- Computer
- Files (COMPUTER ARITHMETIC)
- Demonstration Materials
    - Handouts
    - PowerPoint Presentation
    - Video clips
    - Other Instructional materials

ARITHMETIC

INTRODUCTION

- A basic operation in all digital computers is the addition or subtraction of two numbers.
- Arithmetic operations occur at the machine instruction level.
- They are implemented, along with basic logic functions such as AND, OR, NOT, and EXCLUSIVE-OR, in the arithmetic and logic unit (ALU) subsystem of the processor.
- The time needed to perform an addition operation affects the processor's performance.
- Multiply and divide operations, which require more complex circuitry than either addition or subtraction operations, also affect performance.

- Compared with arithmetic operations, logic operations are simple to implement using combinational circuitry. They require only independent Boolean operations on individual bit positions on the operands, whereas carry/borrow lateral signals are required in arithmetic operations

REVIEW OF NUMBER REPRESENTATIONS

- Three systems are widely used for representing both positive and negative numbers:
  1. Sign and Magnitude
  2. 1's – complement
  3. 2's – complement
- For all three systems, the leftmost bit is 0 for positive numbers and 1 is for negative numbers
- Binary, Signed Integer Representations

$b_3\ b_2\ b_1\ b_0$	Sign and Magnitude	**1's** Complement	**2's** Complement
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	- 0	- 7	- 8
1001	- 1	- 6	- 7
1010	- 2	- 5	- 6
1011	- 3	- 4	- 5
1100	- 4	- 3	- 4
1101	- 5	- 2	- 3
1110	- 6	- 1	- 2
1111	- 7	- 0	- 1

- Positive values have identical representations in all systems.
- In the sign-and-magnitude system, negative values are represented by changing the MSB of the positive representation to 1.

  Example:

+ 5 = 0101

- 5 = 1101

- In the 1's-complement system, negative values are obtained by complementing each bit of the corresponding positive representation.

  Example:

  +3 = 0011

  -3 = 1100

- Clearly, the same operation, bit complementing, is done in converting a negative number to the corresponding positive value.

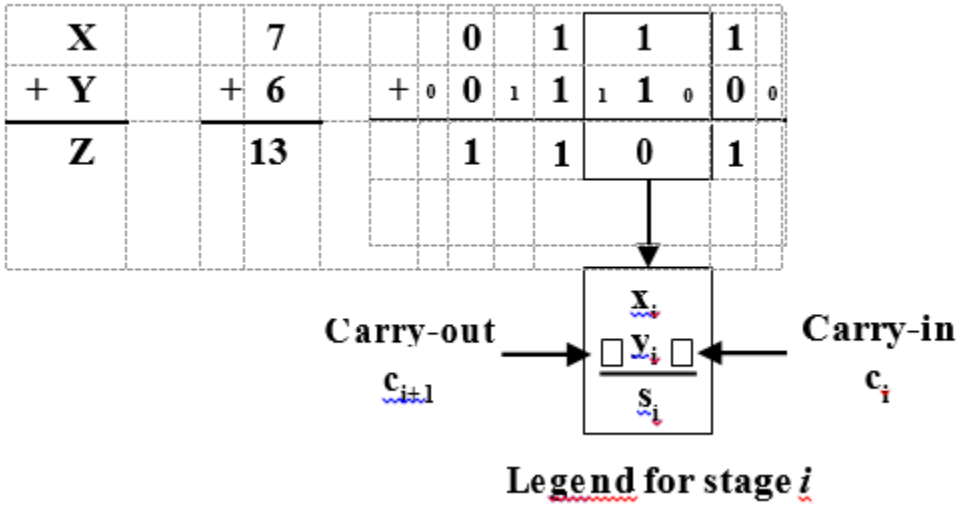
ADDITION OF POSITIVE NUMBERS

Addition of 1-bit numbers

0	1	0	1
+ 0	+ 0	+ 1	+ 1
0	1	1	1 0

Carry-out

- The sum of 1 and 1 requires the 2-bit vector 1 0 to represent the value 2.
- In order to add multiple-bit numbers, bit pairs are added starting from the low-order (right) end of the bit vectors, propagating carries toward the high-order (left) end.



- The truth table for the sum and carry-out functions for adding two equally weighted bits x_i and y_i:

xi	yi	ci	si	ci+1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

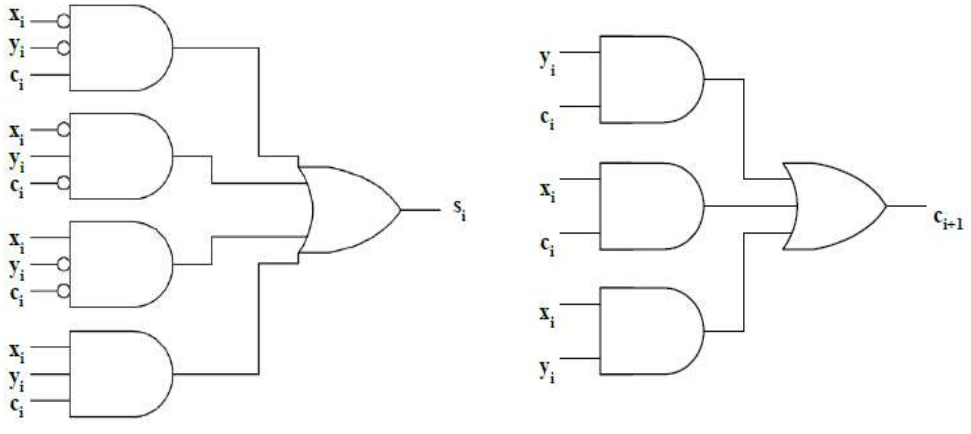
- The logic equation for the sum bit s_i is given by:

$$s_i = \overline{x_i}\ y_i\ \overline{c_i} + x_i\ \overline{y_i}\ \overline{c_i} + x_i\ y_i\ c_i + \overline{x_i}\ \overline{y_i}\ c_i$$

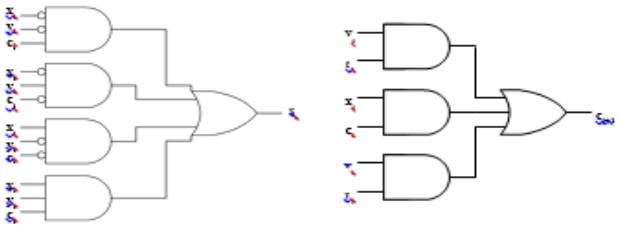
- The logic equation for the carry-out bit c_{i+1} is given by:

$$c_{i+1} = y_i\ c_i + x_i\ c_i + x_i\ y_i$$

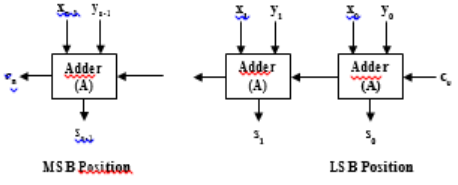
- A straightforward, 2-level, combinational logic circuit implementation of the truth table for addition is shown below:



- A straightforward, 2-level, combinational logic circuit implementation of the truth table for addition is shown below:

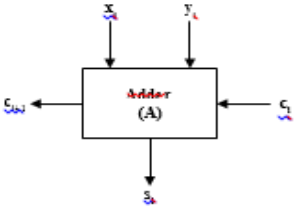


- A cascaded connection of *n* adder blocks can be used to add two *n*-bit numbers.

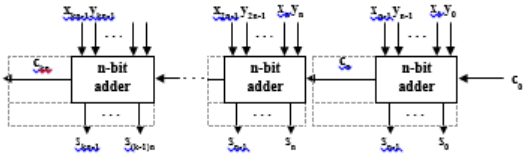


Since the carries must propagate, or ripple, through this cascade, the configuration is called an *n-bit ripple-carry adder*.

- Simplified block representation of an adder (sometimes called a *full-adder*)



- A cascade of *k n*-adders:



DESIGN OF FAST ADDERS

- The *n*-bit ripple-carry adder may have too much delay in developing its outputs, *s0* through *sn-1* and *cn*.
- Suppose that the delay from *ci* to *ci+1* of any Adder block is 1 ns (assuming a gate delay of 0.5 ns). An *n*-bit addition can be performed in the time it takes the carry signal to reach the *cn-1* position plus the time it takes to develop *sn-1*. Assuming the last delay is 1.5 ns, a 32-bit addition takes (31 □ 1 ns) + 1.5 ns = 32.5 ns.
- Two approaches can be taken to reduce this delay to the desired 10-ns range. The first approach is to use faster electronic circuit technology in implementing the ripple-carry logic design. The second approach is to use a different logic network structure.

Logic structures for fast adder design must speed up the generation of the carry signals. The logic expressions for *si* (sum) and *ci+1* (carry-out) of stage *i* are:

$$s_i = \overline{x}_i \overline{y}_i c_i + \overline{x}_i y_i \overline{c}_i + x_i \overline{y}_i \overline{c}_i + x_i y_i c_i$$

$$c_{i+1} = x_i y_i + y_i c_i + x_i c_i$$

By doing the following:

$$c_{i+1} = x_i y_i + (y_i + x_i) c_i$$

Let $G_i = x_i y_i$ and $P_i = x_i + y_i$

$$c_{i+1} = G_i + P_i c_i$$

- The expressions G_i and P_i are called the *generate* and *propagate* functions for stage *i*. Take note that both functions are dependent only on the *X* and *Y* inputs and not on any carry input.

- Expanding c_i in terms of *i*–1 subscripted variables:

$$c_i = x_{i-1} y_{i-1} + (y_{i-1} + x_{i-1}) c_{i-1}$$

$$c_i = G_{i-1} + P_{i-1} c_{i-1}$$

- Substituting into the c_{i+1} equation:

$$c_{i+1} = G_i + P_i c_i$$

$$c_{i+1} = G_i + P_i [G_{i-1} + P_{i-1} c_{i-1}]$$

$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} c_{i-1}$$

- Continuing this type of expansion, the final expression for any carry variable is:

$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \ldots +$$

$$P_i P_{i-i} \ldots P_1 G_0 + P_i P_{i-1} \ldots P_0 c_0$$

- Thus all carries can be obtained three-logic delays after the input operands *X*, *Y*, and c_0 are available, because only one gate delay is needed to develop all P_i and G_i signals, followed by two gate delays in the AND-OR circuit for

ci+1.

- After another three gate delays (one delay to invert each carry and two further delays to form each sum bit as shown earlier), all sum bits are available.

- Therefore, independent of *n*, the *n*-bit addition process requires only six levels of logic.

Example:

Assume a 4-bit adder:

$$C1 = G0 + P0C0$$

$$C2 = G1 + P1G0 + P1P0C0$$

$$C3 = G2 + P2G1 + P2P1G0 + P2P1P0C0$$

$$C4 = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0C0$$

- It will take 1 gate delay (0.5 ns) to produce all the generate and propagate functions after the availability of the *X* and *Y* inputs.

- After the generate and propagate functions become available, it will take two gate delays (1 ns) to produce all the carry signals (*c1*, *c2*, *c3*, *c4*).

- After all the carry signals become available, it will take three gate delays (1.5 ns) to produce all the sum bits.

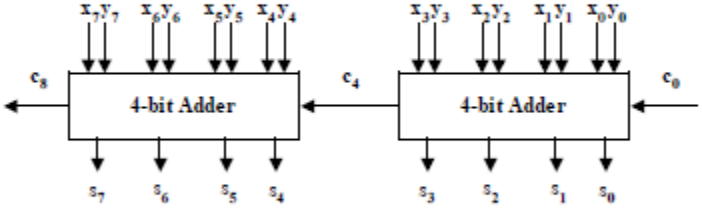
- Therefore the total time for a 4-bit addition will take only 3 ns. It also takes the same time to perform any *n*-bit addition.
- Fast adders that form carry functions are called *carry-lookahead adders*.

- A practical problem with the carry-lookahead approach is the gate fan-in constraints. The expression for *ci+1* requires *i* + 2 inputs to the largest AND term and *i* + 2 inputs to the OR term.

Consider the addition of two 8-bit numbers. For the ripple-carry adder, the total time for an 8-bit addition is (7 □ 1 ns) + 1.5 ns = 8.5 ns. For the carry-lookahead adder, the total time is 3 ns. However, the carry-lookahead adder requires a fan-in of nine for the basic gates.

- A solution to this problem is to implement the carry-lookahead technique on a per block basis.

For the 8-bit addition:



The carries inside each 4-bit adder block are formed by lookahead circuits. However, they still ripple between blocks.

Because of the lookahead circuits in the rightmost adder, c_4 will be formed after 3 gate delays (1.5 ns).

After c_4 becomes available, the leftmost block will produce the remaining part of the sum in 2.5 ns (because of the lookahead circuits). Therefore, the total time for an 8-bit addition is 4.0 ns.

SIGNED ADDITION AND SUBTRACTION

- Out of the three methods of representing signed numbers, the 2's complement representation is the best method in terms of efficiently implementing addition and subtraction.
- he rules for governing the addition and subtraction of *n*-bit signed numbers using the 2's complement representation system are:
  1. To *add* two numbers, add their representations in an *n*-bit adder, ignoring the carry-out signal from the MSB position. The sum will be the algebraically correct value in 2's complement representation as long as the answer is in the range –2n-1 through +2n-1 – 1.
  2. To *subtract* two numbers *X* and *Y*, that is to perform *X* – *Y*, form the 2's complement of *Y* and then add it to *X*, as in rule 1. Again, the result will be the algebraically correct value in 2's complement representation as long as the answer is in the range –2n-1 through +2n-1 – 1.

Examples:

$$\begin{array}{r} 0010 \\ +\ 0011 \\ \hline 0101 \end{array} \quad \begin{array}{r} (+2) \\ +\ (+3) \\ \hline (+5) \end{array} \qquad \begin{array}{r} 0100 \\ +\ 1010 \\ \hline 1110 \end{array} \quad \begin{array}{r} (+4) \\ +\ (-6) \\ \hline (-2) \end{array}$$

$$\begin{array}{r} 1011 \\ +\ 1110 \\ \hline 1001 \end{array} \quad \begin{array}{r} (-5) \\ +\ (-2) \\ \hline (-7) \end{array} \qquad \begin{array}{r} 0111 \\ +\ 1101 \\ \hline 0100 \end{array} \quad \begin{array}{r} (+7) \\ +\ (-3) \\ \hline (+4) \end{array}$$

$$\begin{array}{r} 0010 \\ -\ 0100 \\ \hline \end{array} \quad \begin{array}{r} (+2) \\ -\ (+4) \\ \hline \end{array} \Rightarrow \begin{array}{r} 0010 \\ +\ 1100 \\ \hline 1110 \end{array} \quad \begin{array}{r} \\ \hline (-2) \end{array}$$

$$\begin{array}{r} 0110 \\ -\ 0011 \\ \hline \end{array} \quad \begin{array}{r} (+6) \\ -\ (+3) \\ \hline \end{array} \Rightarrow \begin{array}{r} 0110 \\ +\ 1101 \\ \hline 0011 \end{array} \quad \begin{array}{r} \\ \hline (+3) \end{array}$$

$$\begin{array}{r} 1001 \\ -\ 1011 \\ \hline \end{array} \quad \begin{array}{r} (-7) \\ -\ (-5) \\ \hline \end{array} \Rightarrow \begin{array}{r} 1001 \\ +\ 0101 \\ \hline 1110 \end{array} \quad \begin{array}{r} \\ \hline (-2) \end{array}$$

$$\begin{array}{r} 1001 \\ -\ 0001 \\ \hline \end{array} \quad \begin{array}{r} (-7) \\ -\ (+1) \\ \hline \end{array} \Rightarrow \begin{array}{r} 1001 \\ +\ 1111 \\ \hline 1000 \end{array} \quad \begin{array}{r} \\ \hline (-8) \end{array}$$

$$\begin{array}{r} 0010 \\ -\ 1101 \\ \hline \end{array} \quad \begin{array}{r} (+2) \\ -\ (-3) \\ \hline \end{array} \Rightarrow \begin{array}{r} 0010 \\ +\ 0011 \\ \hline 0101 \end{array} \quad \begin{array}{r} \\ \hline (+5) \end{array}$$

$$\begin{array}{r} 1101 \\ -\ 1001 \\ \hline \end{array} \quad \begin{array}{r} (-3) \\ -\ (-7) \\ \hline \end{array} \Rightarrow \begin{array}{r} 1101 \\ +\ 0111 \\ \hline 0100 \end{array} \quad + \begin{array}{r} \\ \hline (+4) \end{array}$$

MULTIPLICATION OF POSITIVE NUMBERS

- The usual algorithm for multiplying integers by hand for the binary system:

```
      1 1 0 1   (13) Multiplicand M
  x   1 0 1 1   (11) Multiplier Q
  -------------
      1 1 0 1
    1 1 0 1
  0 0 0 0
1 1 0 1
-------------
1 0 0 0 1 1 1 1   (143) Product P
```

- This algorithm applies to unsigned numbers and to positive numbers.

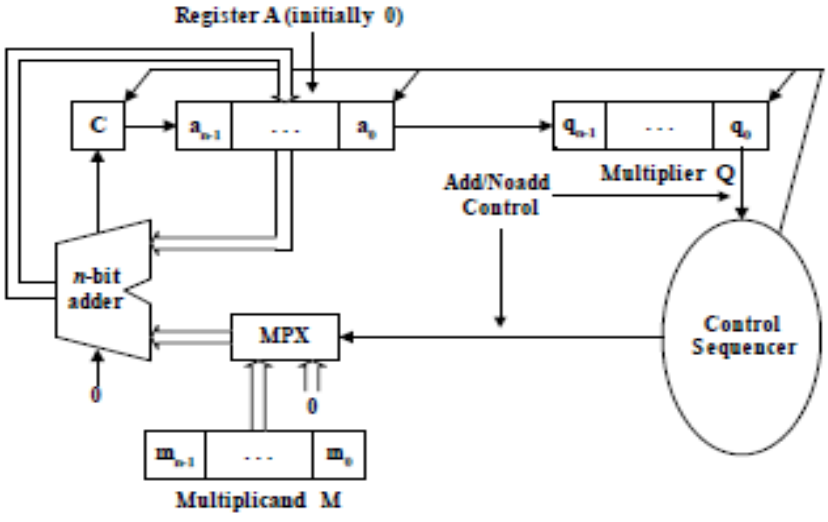
- The product of two n-digit numbers can be accommodated in $2n$ digits, so the product of two 4-bit numbers fits into 8 bits.

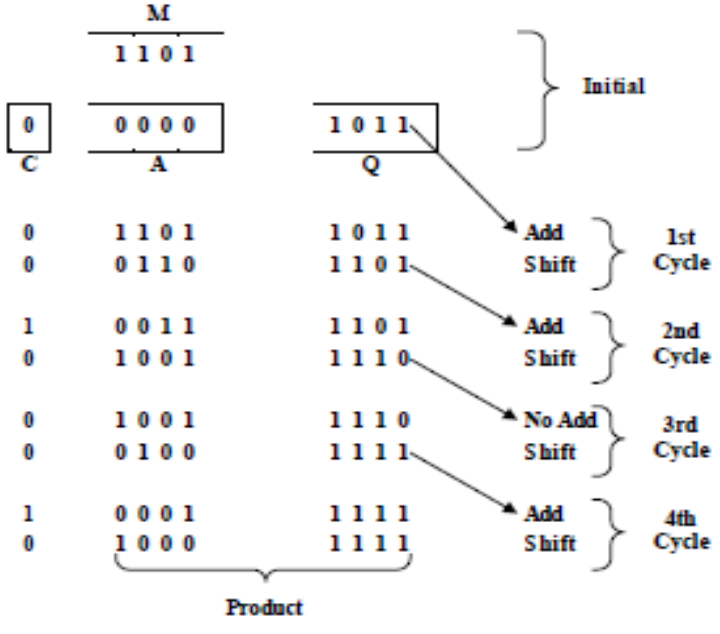
- In the binary system, multiplication of the multiplicand by one bit of the multiplier is easy.

  If the multiplier bit is 1, the multiplicand is entered in the appropriate position to be added to the partial product. If the multiplier is 0, then 0s are entered.

- In early computers, because of logic costs, the adder circuitry in the ALU was used to perform multiplication sequentially.

☞ Block diagram for the sequential circuit binary multiplier:





SIGNED-OPERAND MULTIPLICATION

- Consider first the case of a positive multiplier and a negative multiplicand. In adding a negative multiplicand to a partial product, it is necessary to extend the sign-bit value of the multiplicand to the left as far as the product will extend.

Example:

					1	0	0	1	1	(-13)
				×	0	1	0	1	1	(+11)
1	*1*	*1*	*1*	*1*	1	0	0	1	1	
1	*1*	*1*	*1*	1	0	0	1	1		
0	*0*	*0*	0	0	0	0	0			
1	*1*	1	0	0	1	1				
0	0	0	0	0	0					
1	1	0	1	1	1	0	0	0	1	(-143)

- For a negative multiplier, a straightforward solution is to form the 2's complement of both the multiplier and the multiplicand and proceed as in the case of a positive multiplier.

  This is possible because complementation of both operands does not change the value or the sign of the product.

☞ *Booth Algorithm*

A powerful algorithm for signed-number multiplication, the Booth algorithm generates a $2n$-bit product and treats both positive and negative numbers uniformly.

Consider a multiplication operation in which the multiplier is positive and has a single block of 1s, such as 0011110. To derive the product, one can add four appropriately shifted versions of the multiplicand, as in the standard procedure.

However, the number of required operations can be reduced by regarding the multiplier as the difference between two numbers:

$$\begin{array}{rl} & 0100000\,(32) \\ - & 0000010\,(2) \\ \hline & 0011110\,(30) \end{array}$$

This suggests that the product can be generated by adding 2^5 times the multiplicand to the 2's complement of 2^1 times the multiplicand.

For convenience, the sequence of required operations can be described by recording the preceding multiplier as 0 +1 0 0 0 –1 0.

Normal multiplication algorithm:

```
          0 1 0 1 1 0 1
          0 0 1 1 1 1 0
          0 0 0 0 0 0 0
        0 1 0 1 1 0 1
      0 1 0 1 1 0 1
    0 1 0 1 1 0 1
  0 1 0 1 1 0 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
-----------------------
0 0 0 1 0 1 0 1 0 0 0 1 1 0
```

Booth multiplication algorithm:

```
          0 1 0 1 1 0 1
          0+1 0 0 -1 0
-----------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 0 1 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 1 0 1
0 0 0 0 0 0 0 0
-----------------------
0 0 0 1 0 1 0 1 0 0 0 1 1 0
```

The Booth algorithm can also be used for negative numbers.

```
        0 1 1 0 1     (+13)
x       1 1 0 1 0     (-6)
```

```
          0 1 1 0 1
          0-1+1-10
-----------------------
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 1 1
0 0 0 0 1 1 0 1
1 1 1 0 0 1 1
0 0 0 0 0 0
-----------------------
1 1 1 0 1 1 0 0 1 0     (-78)
```

The transformation 011...110 ⇒ +100...0-10 is called *skipping over 1s*. This term is derived from the case in which the multiplier has its 1s grouped into a few contiguous blocks; only a few versions of the multiplicand, that is, the summands, must be added to generate the product, thus speeding up the multiplication operation.

In general, in the Booth scheme, -1 times the shifted multiplicand is selected when moving from 0 to 1, and +1 times the shifted multiplicand is selected when moving from 1 to 0, as the multiplier is scanned from right to left.

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	0 x M
0	1	+1 x M
1	0	- 1 x M
1	1	0 x M

The Booth algorithm clearly extends to any number of 1s in a multiplier, including the situation in which a single 1 is considered a block.

Example:

```
0 0 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0
                ⇓
0 +1-1 +1 0 -1  0 +1 0  0 -1 +1-1 +1 0 -1  0 0
```

However, in the worst case – that of alternating 1s and 0s in the multiplier – each bit of the multiplier selects a summand. In fact, this results in more summands than if the Booth algorithm were not used.

Examples:

Worst-case Multiplier:

```
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
              ⇓
+1-1 +1 -1+1 -1+1 -1+1-1 +1-1 +1 -1+1-1
```

Ordinary Multiplier:

```
1 1 0 0 0 1 0 1 1 0 1 1 1 1 0 0
              ⇓
0 -1 0 0+1 -1+1 0 -1+1 0 0 0 -1 0 0
```

Good Multiplier:

```
0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1
              ⇓
0 0 0 0+1 0 0 0 0 -1 0 0 0+1 0 -1
```

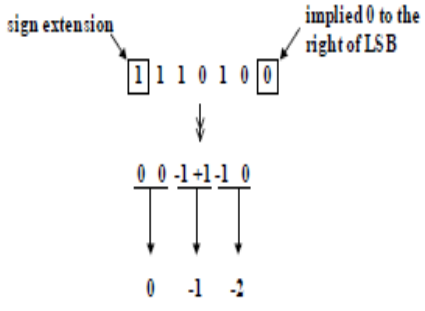
FAST MULTIPLICATION

- A multiplication speedup technique will now be discussed that guarantees at most $n/2$ summands and that uniformly handles the signed-operand case. This is twice as fast as the worst-case Booth algorithm situation.

- Recall that the Booth technique multiplier q_i selects a summand as a function of the bit q_i-1 on its right. The summand selected is shifted i binary positions to the left of the LSB position of the product before the summand is added. Let this summand position as i (SPi).

- The basic idea of the speedup technique is to use the bits i + 1and i to select a summand, as a function of bit i – 1, to be added at SPi. The $n/2$ summands are thus selected by bit pairs (x_1, x_0), (x_3, x_2), (x_5, x_4), and so on. The technique is called the *bit-pair recoding method*.

Example:



Consider the multiplier 11010. Its Booth representation is given as 0 –1 +1 –1 0. By grouping the Booth-recoded selectors, one can obtain a single, appropriately shifted summand for each pair.

The rightmost Booth pair, (-1, 0), is equivalent to –2 x (multiplicand M) at SP0. The next pair, (-1, +1), is equivalent to –1 x M at SP2; finally, the left most pair of zeros is equivalent to 0 x M at SP4.

Restating these selections in terms of the original multiplier bits, the case of (1,0) with 0 on the right selects –2 x M; the case of (1,0) with 1 on the right selects –1 x M; and the case of (1,1) with 1 on the right selects 0 x M.

INTEGER DIVISION

- Longhand division examples:

```
      21                    10101
13 | 274          1101 | 100010010
   26                      1101
    14                     10000
    13                      1101
     1                      1110
                            1101
                               1
```

- A circuit that implements division by this longhand method operates as follows:

  It positions the divisor appropriately with respect to the dividend and performs a subtraction. If the remainder is zero or positive, a quotient bit is determined, the remainder is extended by another bit of the dividend, the divisor is repositioned, and another subtraction is performed.

  On the other hand, if the remainder is negative, a quotient bit of 0 is determined, the dividend is restored by adding back the divisor, and the divisor is repositioned for another subtraction.

- This algorithm can be improved by avoiding the need for restoring A after an unsuccessful subtraction (subtraction is said to be unsuccessful if the result is negative).

  Consider the sequence of operations that takes place after the subtraction operation in the preceding algorithm. If A is positive, we shift left and subtract M; that is, we perform 2A – M. If A is negative, we restore it by performing A + M. and then we shift it left and subtract M. This is equivalent to performing 2A + M. The q_0 bit is appropriately set to 0 or 1 after the correct operation has been performed. All of these can be summarized in the following *nonrestoring- division* algorithm:

  Step 1: Do the following n times:

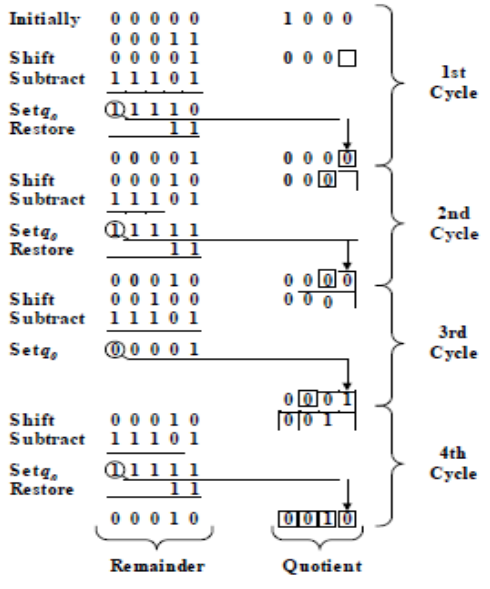
  If the sign of A is 0, shift A and Q one bit position and subtract M from A; otherwise shift A and Q left and add M to A.

  If the sign of A is 0, set q_0 to 1; otherwise, set q_0 to 0.

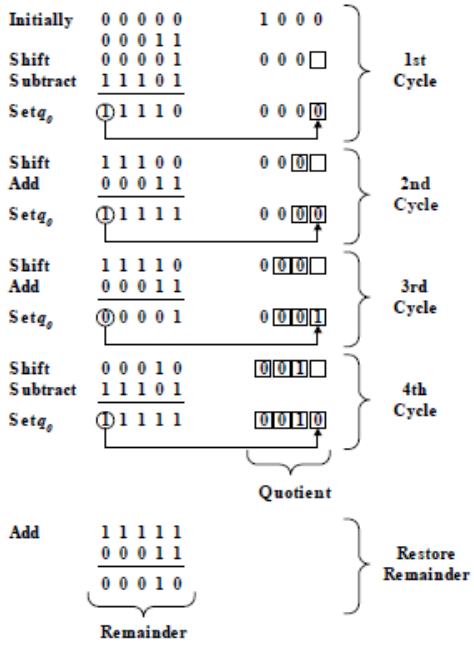
  Step 2: If the sign of A is 1, add M to A.

- A restoring-division example:

```
          1 0
      ------
1 1 | 1 0 0 0
      1 1
      -----
      1 0
```



- A nonrestoring-division example:



Watch:

https://www.youtube.com/watch?v=AtJQftULx8A&list=PLvuOIW-67f9_0cO2mf7uyqK7NLrHt7E-5

Read:

https://www.studytonight.com/computer-architecture/input-output-organisation

https://www.studytonight.com/computer-architecture/array-processor

Activities/Assessments:

1. Have a recap on what have been discussed in this chapter.
2. Perform your own example of the Ripple Carry Adder and Carry Look-Ahead operations.
3. Give at least 1 example of Booth Algorithm, Bit-Pair Recoding techniques, Restoring Division and Non-Restoring Division.
4. The following are the key points:
    a. Addition Algorithm
    b. Subtraction Algorithm
    c. Multiplication Algorithm
    d. Division Algorithm
5. Answer the following:

Exercise No. 2.

Determine the time it takes to compute for the sum using ripple carry adders, carry-lookahead adders, and 4-bit adder blocks in cascade. Assume that the gate delay is 0.6823 ns. Show your solution.

	S & M	1's C	2's C
1. + 300			
2. 1 01101110			
3. - 472			
4. 609			
5. 0 11101011			

Quiz No. 3

MULTIPLE CHOICE [20 points] Choose the letter of the correct answer. Write your answer on the space provided for. If the answer is not found among the choices, write ***RM***. Use ***CAPITAL LETTERS*** only.

_____1. Assuming 8 bits and two's complement are used, the decimal value of 01101101 is:
A. 77 B. 109 C. 146 D. 147

_____2. Assuming 8 bits and one's complement are used, the decimal value of 01101101 is:
A. 77 B. 109 C. 146 D. 147

_____3. Assuming 8 bits and sign-and-magnitude are used, the decimal value of 01101101 is:
A. 77 B. 109 C. 146 D. 147

_____4. Assuming 8 bits and two's complement are used, the decimal value of 11110110 is:
A. -9 B. -10 C. -118 D. -246

_____5. Assuming 8 bits and one's complement are used, the decimal value of 11110110 is:
A. -9 B. -10 C. -118 D. -246

_____6. Assuming 8 bits and sign-and-magnitude are used, the decimal value of 11110110 is:
A. -9 B. -10 C. -118 D. -246

_____7. Assuming 9 bits and two's complement are used, the binary representation of the decimal –127 is:
A. 110000000 B. 110000001 C. 101111111 D. 101111110

_____8. Assuming 9 bits and one's complement are used, the binary representation of the decimal –127 is:
A. 110000000 B. 110000001 C. 101111111 D. 101111110

_____9. Assuming 9 bits and sign-and-magnitude are used, the binary representation of the decimal –127 is:
A. 110000000 B. 110000001 C. 101111111 D. 101111110

_____10. Assuming 9 bits and two's complement are used, the binary representation of the decimal –256 is:
A. 100000000 B. 1101111111 C. 111111111 D. 101111110

_____11. Assuming 9 bits and one's complement are used, the binary representation of the decimal –256 is:
A. 100000000 B. 1101111111 C. 111111111 D. 101111110

_____12. Assuming 9 bits and sign-and-magnitude are used, the binary representation of the decimal –256 is:
A. 100000000 B. 1101111111 C. 111111111 D. 101111110

_____13. Assuming 9 bits and two's complement are used, the binary representation of the decimal +205 is:
A. 000110010 B. 000110011 C. 011001101 D. Out of range

_____14. Assuming 9 bits and one's complement are used, the binary representation of the decimal +205 is:
A. 000110010 B. 000110011 C. 011001101 D. Out of range

_____15. Assuming 9 bits and sign-and-magnitude are used, the binary representation of the decimal +205 is:
A. 000110010 B. 000110011 C. 011001101 D. Out of range

_____16. If there are 15 bits, the highest positive number that can be represented using two's complement is:
A. 16,383 B. 16,384 C. 32,767 D. 32,768

_____17. If there are 15 bits, the highest positive number that can be represented using one's complement is:
A. 16,383 B. 16,384 C. 32,767 D. 32,768

_____18. If there are 15 bits, the highest positive number that can be represented using sign-and-magnitude is:
A. 16,383 B. 16,384 C. 32,767 D. 32,768

_____19. If there are 19 bits, the highest negative number that can be represented using two's complement is:
A. –262,143 B. –262,144 C. –524,287 D. –524,288

_____20. If there are 19 bits, the highest negative number that can be represented using one's complement is:
A. –262,143 B. –262,144 C. –524,287 D. –524,288

CHAPTER 6 - MEMORY SYSTEM

Course Overview:

This course includes the study of Microcomputer Memory, Characteristics of Memory System, Memory Hierarchy, Internal and External Memory, Cache Memory, Elements of Cache Design and Cache Mapping Functions of Computer Systems.

Learning Objectives:

After successful completion of this lesson, you should be able to:

- Discuss the hierarchy of memory storage
- Discuss the characteristics of memory systems
- Differentiate Internal from External memory
- Discuss the operation of cache memory
- Identify the elements of Cache design

Course Materials:

- Computer
- Files (MEMORY SYSTEM)
- Demonstration Materials
    - Handouts
    - PowerPoint Presentation
    - Video clips
    - Other Instructional materials

THE MEMORY SYSTEM

INTRODUCTION

- Programs and the data they operate on are held in the memory of the computer.
- The execution speed of programs is highly dependent on the speed with which instructions and data can be transferred between the processor and the memory.
- It is also important to have a large memory to facilitate the execution of programs that are large and deal with huge amounts of data.

- Ideally, the memory would be fast, large and inexpensive. Unfortunately, it is impossible to meet all three of these requirements simultaneously. Increase in speed and size are achieved at increased cost.
- Much work has gone into developing clever structures that improve the apparent speed and size of the memory, yet keep the cost reasonable.

SOME BASIC CONCEPTS

- The addressing scheme of any computer system determines the maximum size of the main memory that a computer can use.

  Examples:

  1. If number of address bits = 16

     $$\text{MM Size} = 2^{16} = 65{,}536$$
     $$= 64\text{K memory locations}$$

  2. If number of address bits = 32

     $$\text{MM Size} = 2^{32} = 42{,}94{,}967{,}296$$
     $$= 4\text{G memory locations}$$

  3. If number of address bits = 40

     $$\text{MM Size} = 2^{40} = 1{,}099{,}511{,}627{,}776$$
     $$= 1\text{T memory locations}$$

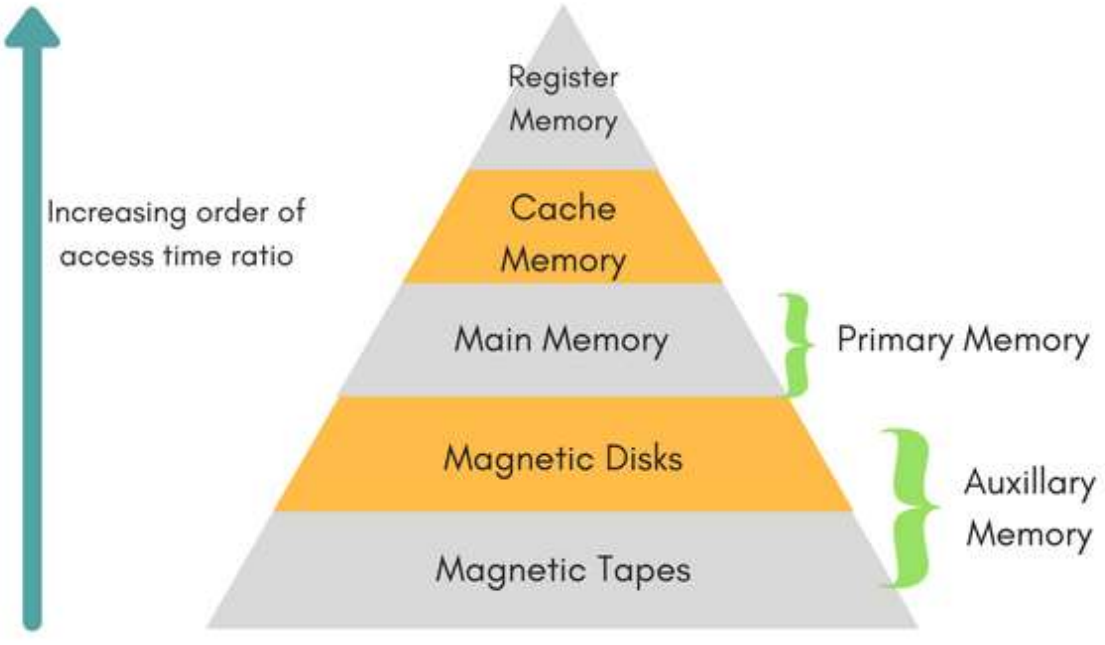
- The MM can usually store and retrieve data in word length quantities. However, the processor can access individual memory bytes as long as each byte has a unique address.

Memory Organization in Computer Architecture

A memory unit is the collection of storage units or devices together. The memory unit stores the binary information in the form of bits. Generally, memory/storage is classified into 2 categories:

- **Volatile Memory**: This loses its data, when power is switched off.
- **Non-Volatile Memory**: This is a permanent storage and does not lose any data when power is switched off.

Memory Hierarchy



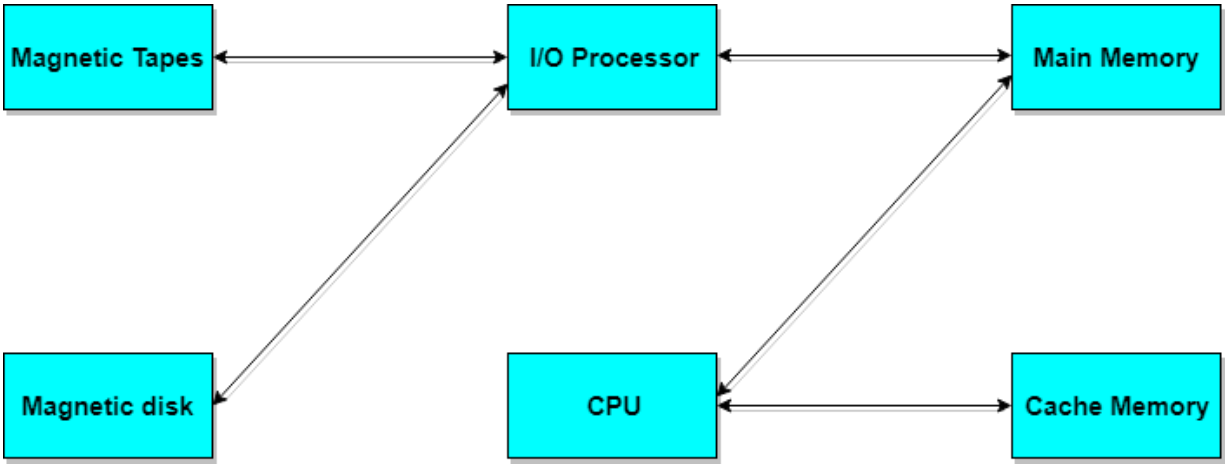
The total memory capacity of a computer can be visualized by hierarchy of components. The memory hierarchy system consists of all storage devices contained in a computer system from the slow Auxiliary Memory to fast Main Memory and to smaller Cache memory.

Auxillary memory access time is generally **1000 times** that of the main memory, hence it is at the bottom of the hierarchy.

The **main memory** occupies the central position because it is equipped to communicate directly with the CPU and with auxiliary memory devices through Input/output processor (I/O).

When the program not residing in main memory is needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space in main memory for other programs that are currently in use.

The **cache memory** is used to store program data which is currently being executed in the CPU. Approximate access time ratio between cache memory and main memory is about **1 to 7~10**



Memory Access Methods

Each memory type, is a collection of numerous memory locations. To access data from any memory, first it must be located and then the data is read from the memory location. Following are the methods to access information from memory locations:

1. **Random Access**: Main memories are random access memories, in which each memory location has a unique address. Using this unique address any memory location can be reached in the same amount of time in any order.
2. **Sequential Access**: This methods allows memory access in a sequence or in order.
3. **Direct Access**: In this mode, information is stored in tracks, with each track having a separate read/write head.

Main Memory

The memory unit that communicates directly within the CPU, Auxillary memory and Cache memory, is called main memory. It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations. Main memory is made up of **RAM** and **ROM**, with RAM integrated circuit chips holing the major share.

- RAM: Random Access Memory
    - **DRAM**: Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10~100 ms. It is slower and cheaper than SRAM.
    - **SRAM**: Static RAM, has a six transistor circuit in each cell and retains data, until powered off.
    - **NVRAM**: Non-Volatile RAM, retains its data, even when turned off. Example: Flash memory.

- ROM: Read Only Memory, is non-volatile and is more like a permanent storage for information. It also stores the **bootstrap loader** program, to load and start the operating system when computer is turned on. **PROM**(Programmable ROM), **EPROM**(Erasable PROM) and **EEPROM**(Electrically Erasable PROM) are some commonly used ROMs.

Auxiliary Memory

Devices that provide backup storage are called auxiliary memory. **For example:** Magnetic disks and tapes are commonly used auxiliary devices. Other devices used as auxiliary memory are magnetic drums, magnetic bubble memory and optical disks.

It is not directly accessible to the CPU, and is accessed using the Input/Output channels.

Cache Memory

The data or contents of the main memory that are used again and again by CPU, are stored in the cache memory so that we can easily access that data in shorter time.

Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory then the CPU moves onto the main memory. It also transfers block of recent data into the cache and keeps on deleting the old data in cache to accommodate the new one.

Hit Ratio

The performance of cache memory is measured in terms of a quantity called **hit ratio**. When the CPU refers to memory and finds the word in cache it is said to produce a **hit**. If the word is not found in cache, it is in main memory then it counts as a **miss**.

The ratio of the number of hits to the total CPU references to memory is called hit ratio.

Hit Ratio = Hit/(Hit + Miss)

Associative Memory

It is also known as **content addressable memory (CAM)**. It is a memory chip in which each bit position can be compared. In this the content is compared in each bit cell which allows very fast table lookup. Since the entire chip can be compared, contents are randomly stored without considering addressing scheme. These chips have less storage capacity than regular memory chips.

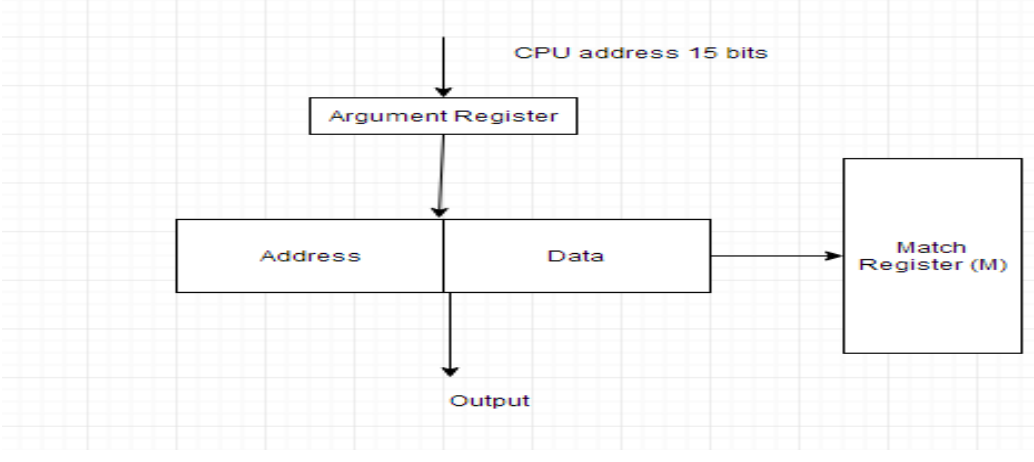
Memory Mapping and Concept of Virtual Memory

The transformation of data from main memory to cache memory is called mapping. There are 3 main types of mapping:

- Associative Mapping
- Direct Mapping
- Block-Set Associative Mapping

Associative Mapping

The associative memory stores both address and data. The address value of 15 bits is 5 digit octal numbers and data is of 12 bits word in 4 digit octal number. A CPU address of 15 bits is placed in **argument register** and the associative memory is searched for matching address.



Example:

A 16 MB main memory is to be upgraded by adding a cache memory of 128 KB. Both the main memory and the cache memory are partitioned into blocks of 512 bytes. How many bits make up the tag and word fields of a memory address?

Solution:

No. of MM Blocks = 16 MB/512

= 32,768

No. of Cache Blocks = 128 KB/512

= 256

No. of Tag Bits = log 32,768 / log 2

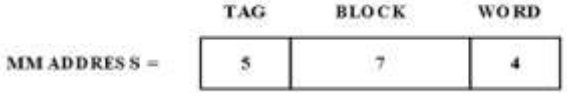
= 15 bits

No. of Word Bits = log 512 / log 2

= 9 bits

Direct Mapping

The CPU address of 15 bits is divided into 2 fields. In this the 9 least significant bits constitute the **index** field and the remaining 6 bits constitute the **tag** field. The number of bits in index field is equal to the number of address bits required to access cache memory.



Example:

A 16 MB main memory is to be upgraded by adding a cache memory of 128 KB. Both the main memory and the cache memory are partitioned into blocks of 512 bytes. How many bits make up the tag, block, and word fields of a memory address?

Solution:

No. of MM Blocks = 16 MB/512

= 32,768

No. of Cache Blocks = 128 KB/512

= 256

No. of MM Blocks per Cache Block = 32,768/256

= 128

No. of Tag Bits = log 128 / log 2

= 7 bits

No. of Block Bits = log 256 / log 2

= 8 bits

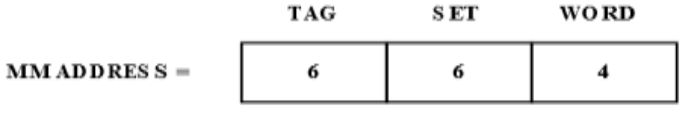
No. of Word Bits = log 512 / log 2

= 9 bits

Block-Set Associative Mapping

The disadvantage of direct mapping is that two words with same index address can't reside in cache memory at the same time. This problem can be overcome by set associative mapping.

In this we can store two or more words of memory under the same index address. Each data word is stored together with its tag and this forms a set.



Example:

A 16 MB main memory is to be upgraded by adding a cache memory of 128 KB. Both the main memory and the cache memory are partitioned into blocks of 512 bytes. Assume that there are 8 blocks to a set. How many bits make up the tag, set, and word fields of a memory address?

Solution:

No. of MM Blocks = 16 MB/512
= 32,768
= 128 KB/512

No. of Cache Blocks = 256

No. of Sets = 256/8
= 32

No. of MM Blocks per Set
= 32,768/32
= 1,024

No. of Tag Bits = log 1,024 / log 2
= 10 bits

No. of Set Bits = log 32 / log 2
= 5 bits

No. of Word Bits = log 512 / log 2
= 9 bits

Replacement Algorithms

Data is continuously replaced with new data in the cache memory using replacement algorithms. Following are the 2 replacement algorithms used:

- FIFO - First in First out. Oldest item is replaced with the latest item.
- LRU - Least Recently Used. Item which is least recently used by CPU is removed.

Virtual Memory

Virtual memory is the separation of logical memory from physical memory. This separation provides large virtual memory for programmers when only small physical memory is available.

Virtual memory is used to give programmers the illusion that they have a very large memory even though the computer has a small main memory. It makes the task of programming easier because the programmer no longer needs to worry about the amount of physical memory available.

Watch:

https://www.youtube.com/watch?v=_kZY4orPQW0

Read:

https://www.studytonight.com/computer-architecture/memory-organization#

Activities/Assessments:

1. Have a recap on what have been discussed in this chapter.
2. Perform your own example of the Ripple Carry Adder and Carry Look-Ahead operations.
3. Give at least 1 example on how to perform Booth Algorithm, Bit-Pair Recoding techniques, Restoring Division and Non-Restoring Division.
4. The following are the key points:
    a. Addition Algorithm
    b. Subtraction Algorithm
    c. Multiplication Algorithm
    d. Division Algorithm
5. Answer the following:

Quiz No. 4

MULTIPLE CHOICE [14 points] Choose the letter of the correct answer. Write your answer on the space provided for. If the answer is not found among the choices, write ***RM***. Use ***CAPITAL LETTERS*** only.

A 16 MB main memory is to be upgraded by adding a cache memory of 256 KB. Both the main memory and the cache memory are partitioned into blocks of 32KB.

500

P 1. How many blocks does the main memory contain?
A. 128 B. 256 C. 512 D. None of the Above

A 2. How many blocks can the cache memory contain?
A. 8 B. 16 C. 32 D. None of the Above

_____3. How many address bits are there?
A. 24 B. 25 C. 26 D. None of the Above

FOR DIRECT MAPPING

_____4. How many bits make up the tag field of a memory address?
A 5 B. 6 C. 7 D. None of the Above

_____5. How many bits make up the block field of a memory address?
A. 5 B. 4 C. 3 D. None of the Above

_____6. How many bits make up the word field of a memory address?
A. 14 B. 15 C. 16 D. None of the Above

_____7. If a main memory word has a hexadecimal address D1A016, to which cache block will it be mapped?
A. Cache Block 3 C. Cache Block 5
B. Cache Block 4 D. None of the Above

_____8. If a main memory word has a hexadecimal address D1A016, what word is being accessed?
A. Word 8214 B. Word 8215 C. Word 8216 D. None of the Above

FOR ASSOCIATIVE MAPPING

_____9. How many bits make up the tag field of a memory address?
A. 8 B. 9 C. 10 D. None of the Above

_____10. How many bits make up the word field of a memory address?
A. 14 B. 15 C. 16 D. None of the Above

FOR PROBLEMS 11 THROUGH 14:

A 32 MB main memory is to be upgraded by adding a cache memory of 256 KB. Both the main memory and the cache memory are partitioned into blocks of 1KB.

FOR BLOCK-SET ASSOCIATIVE MAPPING WITH 4 CACHE BLOCKS/SET

_____11. How many sets are there in the cache?
A. 32 B. 64 C. 128 D. None of the Above

_____12. How many bits make up the set field of a memory address?
A. 6 B. 7 C. 8 D. None of the Above

_____13. If a main memory word has hexadecimal address 123ABCD, to which main memory block does it belong?
A. Block 9480 B. Block 18432 C. Block 18666 D. None of the Above

_____14. If a main memory word has a hexadecimal address 123ABCD, to which cache block will it be mapped?
B. Set 21 B. Set 42 C. Set 64 D. None of the Above

CHAPTER 7 - INPUT-OUTPUT ORGANIZATION

Course Overview:

This course includes the study of Peripheral Devices, I/O Modules, I/O Interface, Modes of Transfer: Programmed, Interrupt Driven and DMA, I/O Processor, Data Communication Processors.

Learning Objectives:

After successful completion of this lesson, you should be able to:

- Discuss the input/output devices that enable communication between computers and the outside world in some form.
- Discuss the reliability of these devices is important.
- Discuss the non-volatile storage mediums, such as disk and flash memory, before learning about mechanisms used to connect the computer to input/output devices.
- Discuss disk system performance measures

Course Materials:

- Computer
- Files (INPUT-OUTPUT ORGANIZATION)
- Demonstration Materials
  - Handouts
  - PowerPoint Presentation
  - Video clips
  - Other Instructional materials
  -

INPUT/OUTPUT ORGANIZATION

Overview

- Computer has ability to exchange data with other devices.
- Human-computer communication
- Computer-computer communication
- Computer-device communication

Accessing I/O Devices

Figure 4.1. A single-bus structure.

Input/Output Subsystem

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input-output operations of the computer system.

Peripheral Devices

Input or output devices that are connected to computer are called **peripheral devices**. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system. These devices are also called **peripherals**.

For example: *Keyboards*, *display units* and *printers* are common peripheral devices.

There are three types of peripherals:

1. **Input peripherals**: Allows user input, from the outside world to the computer. Example: Keyboard, Mouse etc.
2. **Output peripherals**: Allows information output, from the computer to the outside world. Example: Printer, Monitor etc
3. **Input-Output peripherals**: Allows both input (from outside world to computer) as well as, output (from computer to the outside world). Example: Touch screen etc.

Interfaces

Interface is a shared boundary between two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.

There are two types of interface:

1. CPU Interface
2. I/O Interface

Let's understand the I/O Interface in details,

Input-Output Interface

Peripherals connected to a computer need special communication links for interfacing with CPU. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These components are called **input-output interface**

units because they provide communication links between processor bus and peripherals. They provide a method for transferring information between internal system and input-output devices.

Modes of I/O Data Transfer

Data transfer between the central unit and I/O devices can be handled in generally three types of modes which are given below:

1. Programmed I/O
2. Interrupt Initiated I/O
3. Direct Memory Access

Programmed I/O

Programmed I/O instructions are the result of I/O instructions written in computer program. Each data item transfer is initiated by the instruction in the program.

Usually the program controls data transfer to and from CPU and peripheral. Transferring data under programmed I/O requires constant monitoring of the peripherals by the CPU.

Interrupt Initiated I/O

In the programmed I/O method the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is time consuming process because it keeps the processor busy needlessly.

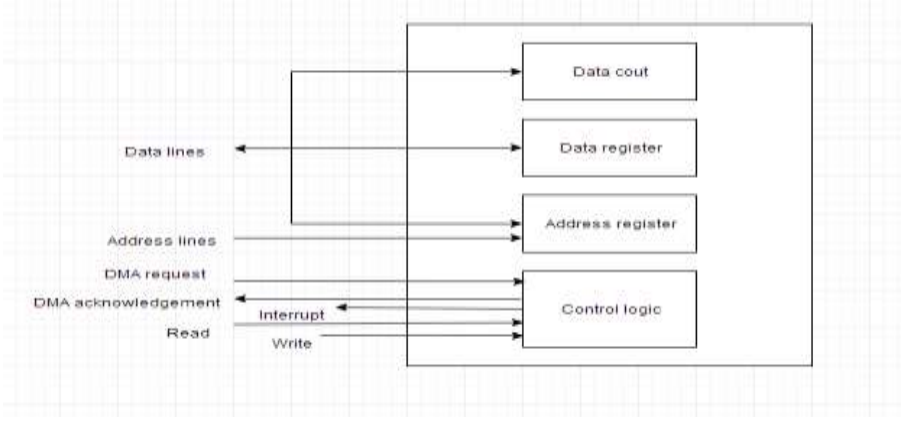
This problem can be overcome by using **interrupt initiated I/O**. In this when the interface determines that the peripheral is ready for data transfer, it generates an interrupt. After receiving the interrupt signal, the CPU stops the task which it is processing and service the I/O transfer and then returns back to its previous processing task.

Direct Memory Access

Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This technique is known as **DMA**.

In this, the interface transfer data to and from the memory through memory bus. A DMA controller manages to transfer data between peripherals and memory unit.

Many hardware systems use DMA such as disk drive controllers, graphic cards, network cards and sound cards etc. It is also used for intra chip data transfer in multicore processors. In DMA, CPU would initiate the transfer, do other operations while the transfer is in progress and receive an interrupt from the DMA controller when the transfer has been completed.



Above figure shows block diagram of DMA

Input/Output Processor

An input-output processor (IOP) is a processor with direct memory access capability. In this, the computer system is divided into a memory unit and number of processors.

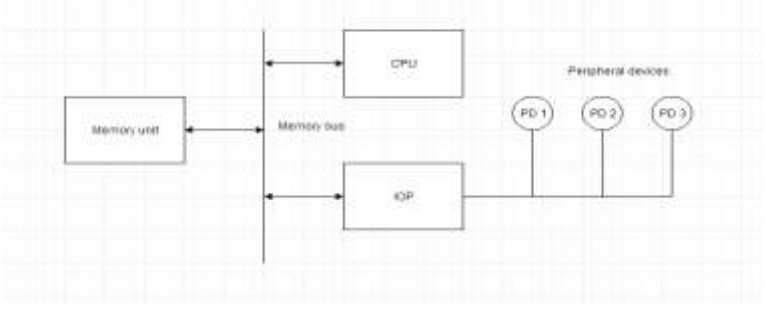
Each IOP controls and manage the input-output tasks. The IOP is similar to CPU except that it handles only the details of I/O processing. The IOP can fetch and execute its own instructions. These IOP instructions are designed to manage I/O transfers only.

Block Diagram of I/O Processor

Below is a block diagram of a computer along with various I/O Processors. The memory unit occupies the central position and can communicate with each processor.

The CPU processes the data required for solving the computational tasks. The IOP provides a path for transfer of data between peripherals and memory. The CPU assigns the task of initiating the I/O program.

The IOP operates independent from CPU and transfer data between peripherals and memory.



The communication between the IOP and the devices is similar to the program control method of transfer. And the communication with the memory is similar to the direct memory access method.

In large scale computers, each processor is independent of other processors and any processor can initiate the operation.

The CPU can act as master and the IOP act as slave processor. The CPU assigns the task of initiating operations but it is the IOP, who executes the instructions, and not the CPU. CPU instructions provide operations to start an I/O transfer. The IOP asks for CPU through interrupt.

Instructions that are read from memory by an IOP are also called *commands* to distinguish them from instructions that are read by CPU. Commands are prepared by programmers and are stored in memory. Command words make the program for IOP. CPU informs the IOP where to find the commands in memory.

Data Communication Processor

A data communication processor is an I/O processor that distributes and collects data from numerous remote terminals connected through telephone and other communication lines to the computer. It is a specialized I/O processor designed to communicate with data communication networks.

Such a communication network consists of variety of devices such as printers, display devices, digital sensors etc. serving many users at once. The data communication processor communicates with each terminal through a single pair of wire. It also communicates with CPU and memory in the same manner as any I/O processor does.

What is Modem?

In a Data Communication Network, the remote terminals are connected to the data communication processor through telephone lines or other wires. Such telephone lines are specially designed for voice communication and computers use them to communicate in digital signals, therefore some conversion is required. These conversions are called modem (modulator-demodulator).

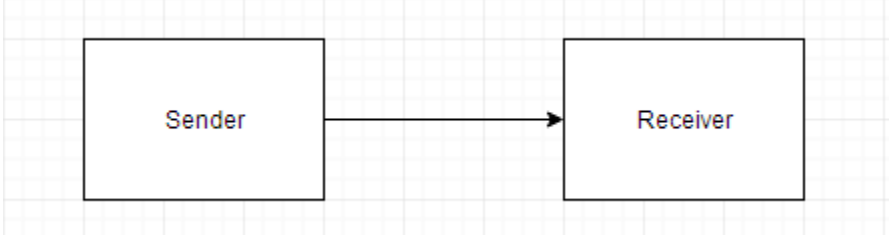
A modem converts digital signal into audio tones to be transmitted over telephone lines and also converts audio tones into digital signal for machine use.

Modes of Transmission

Data can be transmitted between 2 points by three different modes:

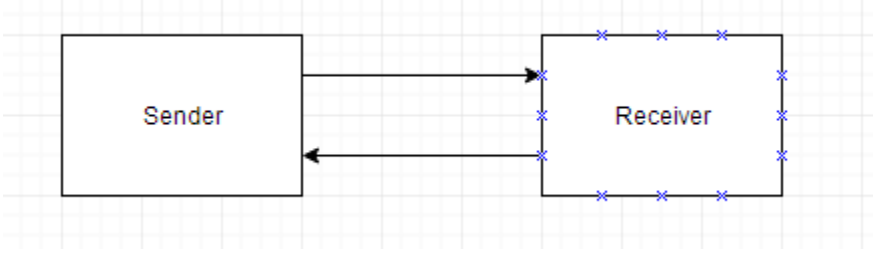
Simplex

A simplex line carries information in one direction only. In this mode receiver cannot communicate with the sender to indicate the occurrence of errors that means only sender can send data but receiver cannot. **For example:** Radio and Television Broadcasting.



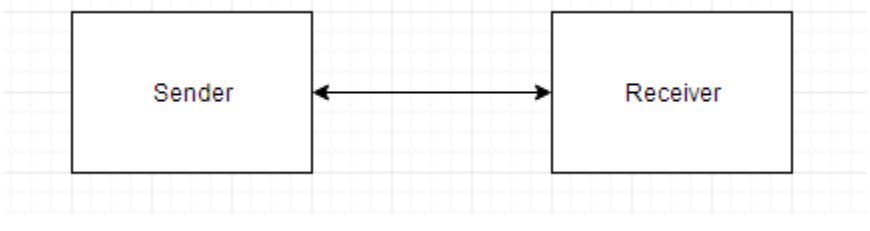
Half Duplex

In half duplex mode, system is capable of transmitting data in both directions but data can be transmitted in one direction only at a time. A pair of wires is needed for this mode. **For example:** Walkie - Talkie.



Full Duplex

In this mode data can be send and received in both directions simultaneously. In this four wire link is used. **For example:** Video Calling, Audio calling etc.



What are Protocols?

The communication lines, modems and other devices used in any transmission are collectively called a **Data Link**. The orderly transmission of data in a data link can be accomplished by a protocol.

A **Protocol** is a set of rules that are followed by interconnecting devices to ensure that all data is passed correctly without any error.

Types of Protocols

There are two types of protocols:

Character Oriented Protocol

It is based on the binary code of character set. The code is mostly used in **ASCII**. It includes upper case and lower case letters, numerals and variety of special symbols. The characters that control the transmission is called **communication control characters**.

Bit Oriented Protocol

It does not use characters in its control field and is independent of any code. It allows the transmission of serial bit stream of any length without the implication of character boundaries.

Watch:

https://www.youtube.com/watch?v=Y17TLZCSe4M

Read:

https://www.studytonight.com/computer-architecture/input-output-processor

Activities/Assessments:

1. Have a recap on what have been discussed in this chapter.
2. Make your own summary of Input/Output Organization
3. Discuss briefly in a pieces of papers the following that are mentioned in the key points.
4. The following are the key points:
    a. Multiprocessor and its Characteristics
    b. Interconnection Structures for Multiprocessor
    c. Inter Processor Communication and Synchronization

CHAPTER 8 - MULTIPROCESSORS AND PARALLEL PROCESSING

Course Overview:

This course includes the study of Multiprocessor and its Characteristics, Interconnection Structures for Multiprocessor and Inter Processor Communication and Synchronization.

Learning Objectives:

After successful completion of this lesson, you should be able to:

- Learn that parallel programming is not easy and that parallel processing imposes certain limitations in performance gains.
- Discuss the concepts of shared memory multiprocessing and cluster processing as two common means of improving performance with parallelism.
- Discuss some of the programming techniques used in the context of parallel machines.

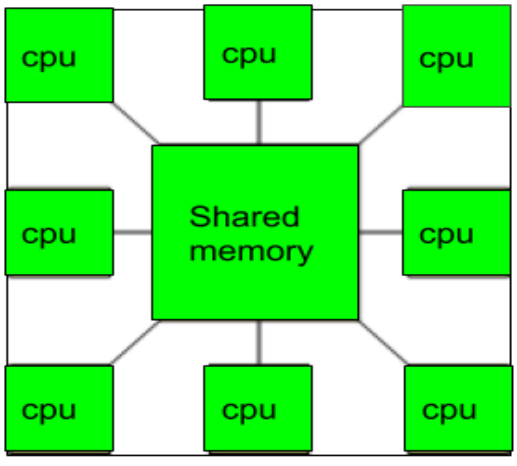
Course Materials:

- Computer
- Files (MULTIPROCESSORS AND PARALLEL PROCESSING)
- Demonstration Materials
    - Handouts
    - PowerPoint Presentation
    - Video clips
    - Other Instructional materials

Multiprocessor

A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM. The main objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.

There are two types of multiprocessors, one is called shared memory multiprocessor and another is distributed memory multiprocessor. In shared memory multiprocessors, all the CPUs shares the common memory but in a distributed memory multiprocessor, every CPU has its own private memory.



Applications of Multiprocessor –

1. As a uniprocessor, such as single instruction, single data stream (SISD).
2. As a multiprocessor, such as single instruction, multiple data stream (SIMD), which is usually used for vector processing.
3. Multiple series of instructions in a single perspective, such as multiple instruction, single data stream (MISD), which is used for describing hyper-threading or pipelined processors.
4. Inside a single system for executing multiple, individual series of instructions in multiple perspectives, such as multiple instruction, multiple data stream (MIMD).

Benefits of using a Multiprocessor –

- Enhanced performance.
- Multiple applications.
- Multi-tasking inside an application.
- High throughput and responsiveness.
- Hardware sharing among CPUs.

Characteristics of Multiprocessors

A multiprocessor system is an interconnection of two or more CPU, with memory and input-output equipment. As defined earlier, multiprocessors can be put under MIMD category. The term multiprocessor is sometimes confused with the term multicomputer. Though both support concurrent operations, there is an important difference between a system with multiple computers and a system with multiple processors. In a multicomputer system, there are multiple computers, with their own operating systems, which communicate with each other, if needed, through communication links. A multiprocessor

system, on the other hand, is controlled by a single operating system, which coordinate the activities of the various processors, either through shared memory or interprocessor messages.

The advantages of multiprocessor systems are:

- Increased reliability because of redundancy in processors
- Increased throughput because of execution of
- multiple jobs in parallel
- portions of the same job in parallel

A single job can be divided into independent tasks, either manually by the programmer, or by the compiler, which finds the portions of the program that are data independent, and can be executed in parallel. The multiprocessors are further classified into two groups depending on the way their memory is organized. The processors with shared memory are called tightly coupled or shared memory processors. The information in these processors is shared through the common memory. Each of the processors can also have their local memory too. The other class of multiprocessors is loosely coupled or distributed memory multi-processors. In this, each processor have their own private memory, and they share information with each other through interconnection switching scheme or message passing.

The principal characteristic of a multiprocessor is its ability to share a set of main memory and some I/O devices. This sharing is possible through some physical connections between them called the interconnection structures.

Parallel Processing and Data Transfer Modes in a Computer System

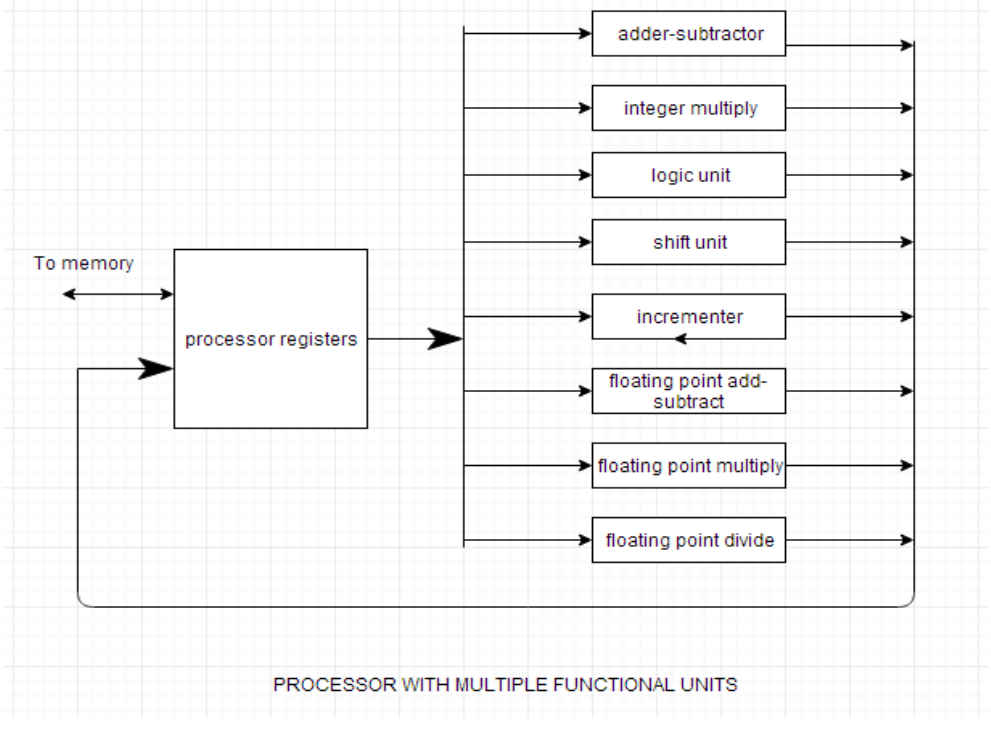
Instead of processing each instruction sequentially, a **parallel processing** system provides concurrent data processing to increase the execution time.

In this the system may have two or more ALU's and should be able to execute two or more instructions at the same time. The purpose of parallel processing is to speed up the computer processing capability and increase its throughput.

NOTE: Throughput is the number of instructions that can be executed in a unit of time.

Parallel processing can be viewed from various levels of complexity. At the lowest level, we distinguish between parallel and serial operations by the type of registers used. At the higher level of complexity, parallel processing can be achieved by using multiple functional units that perform many operations simultaneously.

Data Transfer Modes of a Computer System



PROCESSOR WITH MULTIPLE FUNCTIONAL UNITS

the data transfer mode, computer can be divided into 4 major groups:

1. SISD
2. SIMD
3. MISD
4. MIMD

SISD (Single Instruction Stream, Single Data Stream)

It represents the organization of a single computer containing a control unit, processor unit and a memory unit. Instructions are executed sequentially. It can be achieved by pipelining or multiple functional units.

SIMD (Single Instruction Stream, Multiple Data Stream)

It represents an organization that includes multiple processing units under the control of a common control unit. All processors receive the same instruction from control unit but operate on different parts of the data.

They are highly specialized computers. They are basically used for numerical problems that are expressed in the form of vector or matrix. But they are not suitable for other types of computations

MISD (Multiple Instruction Stream, Single Data Stream)

It consists of a single computer containing multiple processors connected with multiple control units and a common memory unit. It is capable of processing several instructions over single data stream simultaneously. MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

MIMD (Multiple Instruction Stream, Multiple Data Stream

It represents the organization which is capable of processing several programs at same time. It is the organization of a single computer containing multiple processors connected with multiple control units and a shared memory unit. The shared memory unit contains multiple modules to communicate with all processors simultaneously. Multiprocessors and multicomputer are the examples of MIMD. It fulfills the demand of large

Watch:

https://www.youtube.com/watch?v=IZfWjg3U3mA

https://www.youtube.com/watch?v=A5dorbcK78M

https://www.youtube.com/watch?v=q7sgzDH1cR8

Read:

https://engineering.purdue.edu/~ee565/slides/ch4.pdf

Activities/Assessments:

1. Have a recap on what have been discussed in this chapter.
2. Make your own summary of MULTIPROCESSORS AND PARALLEL PROCESSING.
3. Discuss briefly in a pieces of papers the following that are mentioned in the key points.
4. The following are the key points:
    a. Multiprocessor and its Characteristics
    b. Interconnection Structures for Multiprocessor
    c. Inter Processor Communication and Synchronization

REFERENCES:

M. Morris Mano: Computer System Architecture, 2015

William Stalling: Computer organization and architecture, 2016

John P. Hayes: Computer Architecture and Organization, 2017

https://coelms.com

https://www.studytonight.com/computer-architecture/memory-organization#

https://www.studytonight.com/computer-architecture/input-output-organisation

https://www.studytonight.com/computer-architecture/memory-organization#

https://www.studytonight.com/computer-architecture/input-output-organisation

https://www.studytonight.com/computer-architecture/input-output-organisation

Online References:
https://coelms.com

Course Grading System:
To pass this course, one must accumulate at least 75% through the course requirements. The maximum points that a student can obtain through each requirement are shown below.

Requirement/Assessment Task	***Maximum Percentage***
Class Work/Activities/Assignment/Recitation	20%
Midterm/Final Practical Exam / Project	40%
Laboratory Exercises/Experiments and Machine Problems/ Final Project Presentation	40%
	100%

/rlm