UNIT - 1

The Basic Computer Architecture and Data Representation

Lesson Structure

1.0 Objective

1.1 Introduction

1.2 The Von Neumann Architecture

1.3 Instruction Cycle

1.4 Interrupts

1.4.1 Interrupts and Instruction Cycle

1.5 Data Representation

1.6 Number System

1.7 Decimal Representation in Computers

1.8 Alphanumeric Representation

1.9 Error Detection and Correction Codes

1.10 Summary

1.11 Questions

1.10 Suggested Readings

1.0 Objective

After going through this unit we will be able to :

- Define the logical structure of the computer.
- Define the instruction cycle;
- Use binary, octal and hexadecimal numbers;

- Convert decimal numbers to other systems and vice versa;
- Describe the characcter representation in computers;
- Describe the data error checking mechanism and error detection and correction codes.

1.1 Introduction

Information Technology (IT) has become a must for the survival of all business houses with the growing information technology trends. The use of IT is well recognised. Computer is the main component of an Information Technology network. Today, computer technology has it existence in every field starting from railway reservations to medical diagnosis, from TV programmes to satellite launching, from matchmaking to criminal catching. Everywhere we witness the elegance, sophistication and efficiency possible only with the help of computers.

In this unit, we will be introduced to one of the important computer system structures : the **von Neumann Architecture.** In addition, we will be introduced to the concepts of a simple model of Instruction execution.

The computer system is based on the binary system, therefore, we will explain the concepts of binary Data Representation in the Computer System. This unit will introduce us to the number system concepts which include the Binary, Octal, and Hexadecimal notations. In addition, details of various number representations such as floating-point representation, BCD representation and character-based representations have been described in this Unit. Finally the Error detection and correction codes have been described in the Unit.

1.2 The Von Neumann Architecture

John Von Neumann (1903-57) a mathematician was first to publish in 1925 the proposal for a new computer **EDVAC** (Electronic Discrete Variable Program Concept). The EDVAC differed from most of its predecessors in that it stored an processed numbers in true binary or base of **2** form. The von Neumann architecture was the first proposed structure for a general-purpose computer. But before describing the main components of Von Neumann architecture, we will define the term 'computer' as this will help us in discussing about Von Neumann architecture in detail.

Computer is defined as **"An automatic electronic device for making calculations or controlling operations that are expressible in numerical or logical terms".** It is a device that takes some input, processes it and produces the desired output.

The definition clearly categorises computer as an electronic device although the first computers were mechanical and electro-mechanical devices. The definition also points towards the two major areas of computer applications

(a) data processing and

(b) controls or operations

Another important aspect of the definition is the fact that the computer can perform only those operations/calculations, which can be expressed in Logical or Numerical terms.

The basic function performed by a computer is the execution of the program. A program is defined as a sequence of instructions, which operates on data, to perform certain tasks such as finding the greatest of three numbers. The computer controls the execution of the program.

In modern digital computers data is represented in binary form by using two symbols **0** and **1**. These are called Binary digits or bits. But the data which we deal with consists of numeric data and characters such as decimal digits **0** to **9**, alphabets **A** to **Z**, arithmetic operators (e.g.+,-, etc.), relations operators (e.g. =, >, etc.), and many other special characters (e.g.; @, {,], etc.). Therefore, there has to be a mechanism for data representation and data conversion.

Old computers use eight bits to represent a character. This allows up to 28 = 256 different items to be represented uniquely. This collection of eight bits is called a **byte**. Thus, one byte is used to represent one character internally. Most computers use two bytes or four bytes to represent numbers (positive and negative) internally. The data also includes the operational data such as integer, decimal number etc. We will discuss more about data representation in the next section.

Thus, the prime task of a computer is to perform instruction execution. The two most important questions that can be asked here are.

(a) how are the instructions given to the computer ?

(b) how are the instructions interpreted and executed ?

Let us answer the second question first. All computers have a Unit that performs the arithmetic and logical functions. This Unit is called the **Arithmetic Logic Unit (ALU).** But how will the computer determine what operation is to be performed by ALU or in other words who will interpret the operation that is to be performed by ALU ?

This interpretation is done by the **Control Unit** of the computer. The control unit accepts the binary form of instruction and interprets the instruction to generate control signals. These control signals then direct the ALU to perform a specified arithmetic or logic function on the data. Therefore, by changing the control signal

the desired function can be performed on data. Or conversely, the operations that need to be performed on the data can be obtained by providing a set of control signals. Thus, for a new operation one only needs to change the set of control signals.

The unit that interprets a code (a machine instruction) to generate respective control signals is termed as **Control Unit (CU)**. A program now consists of a sequence of codes. Each code is, in effect, an instruction, for the computer. The hardware interprets each of these instructions and generates respective control signals such that the desired operation is performed on the data.

The **Arithmetic Logic Unit (ALU)** and the **Control Unit (CU)** together are termed as the Central Processing Unit (CPU). The CPU is the most important component of a computer's hardware.

All these arithmetic and logical operations are performed in the CPU in special storage areas called registers. The size of the register is one of the important considerations in determining the processing capabilities of the CPU. Register size refers to the amount of information that can be held in a register at a time for processing. The larger the register size, the faster may be the speed of processing.

Now let us answer the first question, how can the instructions and data be put into the computers ? The instructions and data to a computer are given by the external environment. Therefore input devices are needed in the computer to supply data to the computer. The main responsibility of input devices will be to put the data in the form of signals that can be recognized by the system. Similarly, we need another component, which will report the results in proper format. This component is called output device. These components together are referred to as Input/Output (I/O) devices.

In addition, to transfer the information, the computer system internally needs the system interconnections. At present we will not discuss about Input/Output devices and system interconnections in details, except the information that most common input/output devices are keyboard, monitor and printer, and the most common interconnection structure is the Bus structure.

Input devices can bring instructions or data only sequentially. But a program may not be executed sequentially. It may consist of jumps, looping and decision-making instructions. In addition, more than one data element may be required at a time. Therefore, a temporary storage area is needed in a computer to store temporarily the instructions and the data. This component is referred to as memory.

The memory unit stores all the information in a group of memory cells such as a group of 8 binary digits (that is a byte) or 16 bits or 32 bits etc. These groups of memory cells or bits are called memory locations. Each memory location has a

unique address and can be addressed independently. The contents of the desired memory locations are provided to the CPU by referring to the address of the memory location. The amount of information that can be held in the main memory is known as memory capacity. The capacity of the main memory is measured in Mega Bytes (MB) or Giga Bytes (GB). One-kilo byte stands for 2^{10} bytes, which are 1024 bytes (or approximately 1000 bytes). A Mega byte stands for 2^{20} bytes, which is approximately a little over one million bytes, a giga byte is 2^{30} bytes.

Let us now define the key features of von Neumann Architecture :

- The basic function performed by a computer is the execution of a program, which involves :

→ the execution of an instruction, which supplies the information about an operation,

→ and the data on which the operation is to be performed.

The control unit (CU) interprets each of these instructions and generates respective control signals.

- The Arithmetic Logic Unit (ALU) performs the arithmetic and logical operations in special storage area called registers as per the instructions of control unit. The size of the register is one of the important considerations in determining the processing capabilities of the CPU. Register size refers to the amount of information that can be held in a register at a time for processing. The larger the register size, the faster may be the speed of processing.
- An Input/ Output system involving I/O devices allows data input and reporting of the results in proper form and format. For transfer of information a computer system internally needs the system interconnections. One such interconnection structure is BUS interconnection.
- Main memory is needed in a computer to store instructions and the data at the time of Program execution. Memory to CPU is an important data transfer path. The amount of information, which can be transferred between CPU and memory, depends on the size of BUS connecting the two.
- It was pointed out by von-Neumann that the same memory can be used for storing data and instructions. In such a case the data can be treated as data on which processing can be performed, while instructions can be treated as data, which can be used for the generation of control signals.
- The von Neumann machine uses **stored program concept,** i.e., the program and data are stored in the same memory unit for execution. The computers prior to this idea used to store programs and data on separate memories.

Entering and modifying these programs was very difficult as they were entered manually by setting switches, plugging, and unplugging.

- Execution of instructions in von Neumann machine is carried out in a sequential fashion (unless explicitly altered by the program itself) from one instruction to the next.

The block diagram of Von Neumann machine is given below in figure 1.

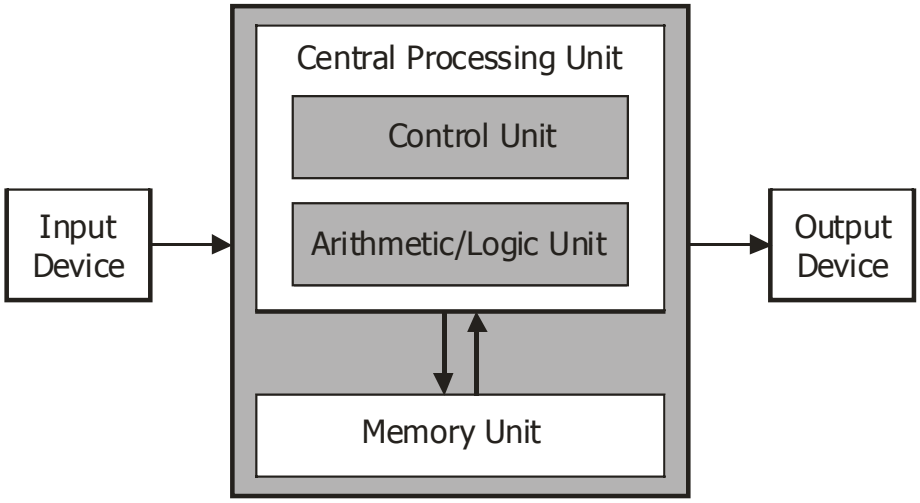


Figure 1 : Von Neumann Architecture

A Von Neumann machine has only a single path between the main memory and control unit (CU). This is referred to as the Von Neumann bottleneck and often limits the performance of the system.

Several other architectures have been suggested for modern computers. We can know about non Von Neumann architectures in suggested readings.

1.3 Instruction Cycle

The sequence of operations performed by the CPU in processing an instruction constitute an instruction cycle. While the details of instruction cycle vary with the type of instruction, all instruction require two major steps.

Step 1 : A fetch during which a new instruction is read from the external memory M and

Step 2 : An execute step during which the operations specified by the instruction are executed.

The flow chart for instruction cycle is given below in the **figure-2,** which shows how the Computer executes a Program, the various types of operations that may be required by computer for execution of program or instruction.

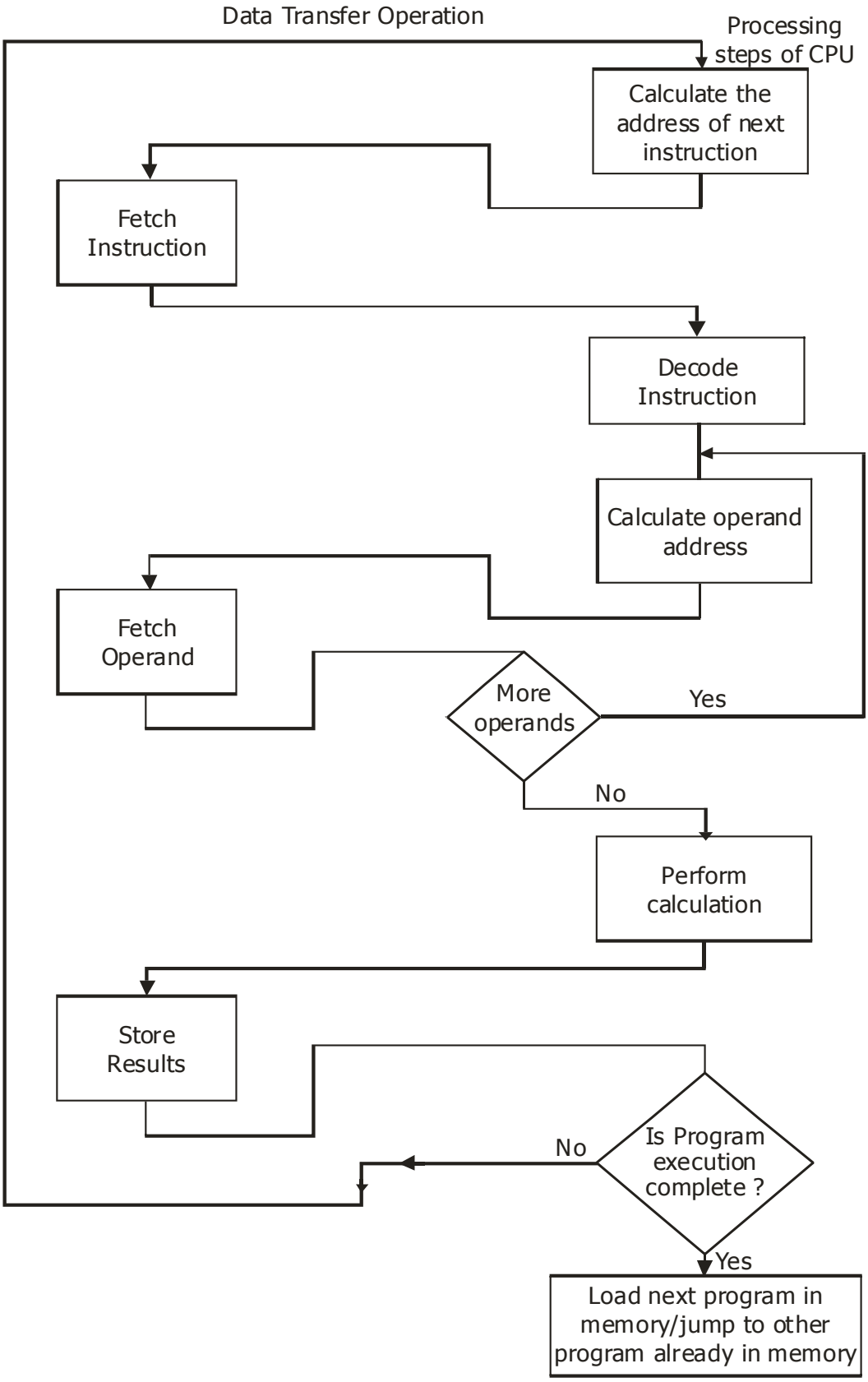


Figure 2 : Instruction Cycle

Thus we can see in the diagram that, the execution cycle for a particular instruction may involve more than one stage and memory references. In addition, an instruction may ask for an I/O operation.

Please note that in the diagram above some steps may be bypassed while some may be visited more than once. The instruction cycle shown in **figure 2** consists of following stages :

- First the address of the next instruction is calculated, based on the size of instruction and memory organization. For example, if in a computer an instruction is of 16 bits and if memory is organized as 16-bits words, then the address of the next instruction is evaluated by adding one in the address of the current instruction. In case, the memory is organized as bytes, which can be addressed individually, then we need to add two in the current instruction address to get the address of the next instruction to be executed in sequence.
- Now, the next instruction is fetched from a memory location to the CPU registers such as instruction register.
- The next state decodes the instruction to determine the type of operation desired and the operands to be used.
- In case the operands need to be fetched from memory or via Input devices, then the address of the memory location or Input device is calculated.
- Next, the operand is fetched (or operands are fetched one by one) from the memory or read from the Input devices.
- Now, the operation, asked by the instruction is performed.
- Finally, the results are written back to memory or Output devices, wherever desired by first calculating the address of the operands and then transferring the values to desired destination.

In a number of cases multiple operands and multiple results are allowed in many computers. An example of such a case may be an instruction MUL X, Y. This instruction requires operand X and Y to be fetched.

In certain machines an operation to be performed may be on an array of numbers or a string of characters. Such an operation involves repeated fetch for the operands without fetching the instruction again, that is, the instruction cycle loops at operand fetch.

Thus, a Program is executed as per the instruction cycle of figure 2. Let us explain this with the help of an example written in C language. C programs is simple to understand.

Problem : Write a program to multiply two numbers.

A sample C program (Assuming two fixed values of numbers as x = 10 any y = 20)

```
1.  #include <stdio.h>
2.  main ()
3.  {
4.  int x = 10, y = 20, product;
5.  product = x*y;
6.  printf ("\n The product value is: % d", product);
7.  }
```

The program at line 4 declares variables that will be equivalent to 3 memory locations namely x, y and product. At line 5 these variables are added and at line 6 the value of **product** is printed. Now let us see how these instructions be executed by CPU.

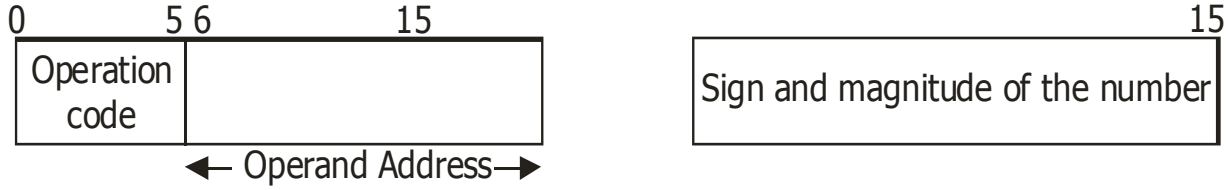
First we need to compile this program to convert it to machine language because C is a high level language.

Let us take an example of an instruction set of a machines of a size of 16 binary digits (bits) instructions and data. Each instruction of the machine consists of two components :

(a) Operation code : that specifies the operation that is to be performed by the instruction,

(b) Address of the operand : in memory on which the given operation is to be performed.

Let us further assume that the size of operation code is assumed to be of six bits. Therefore, rest 10 bits are for the address of the operand. Also the memory word size is assumed to be of 16 bits. **Figure 3.** Shows the instruction and data formats for this machine. However, to simplify our discussion, let us present the operation code using pneumonic like LOAD, ADD, STORE and decimal values of operand addresses and signed decimal values for data.



(a) Instruction format (b) Data format

Figure 3 : Instruction and data format of an assumed machine

The instruction execution is performed in the CPU registers. Let us give details on Registers, the temporary storage location in CPU for program execution. Let us define the minimum set of registers required for von Neumann machines :

Accumulator Register (AC) : This register is to store data temporarily for computation by ALU. AC is considered to contain one of the operands. The result of computation by ALU is also stored back to AC. It implies that the operands value is over-written by the result.

Memory Address Register (MAR) : It specifies the address of memory location from which data or instruction is to be accessed (read operation) or to which the data is to be stored (write operation). Refer to figure 3.

Memory Buffer Register (MBR) : It is a register, which contains the data to be written in the memory (write operation) or it receives the data from the memory (read operation).

Program Counter (PC) : It keeps track of the instruction that is to be executed next, that is, after the execution of an on-going instruction.

Instruction Register (IR) : Here the instructions are loaded prior to execution.

But some time the program is required to terminate in between. The reason of termination may be a program error or an interrupt due to some other reason. We need to answer a question that at what point of time is an interruption to a program execution allowed ? To answer this questions, let us discuss the process used in computer that is called interrupt handling.

1.4 Interrupts

The term interrupt is an exceptional event that causes CPU to temporarily transfer its control from currently executing program to a different program which provides service to the exceptional event. In other words **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. It is like asking a question in a class. When we ask a question in a class by raising hands, the teacher who is explaining some point may respond to our request only after completion of his/her point. Similarly, an interrupt is acknowledged by the CPU when it has completed the currently executing instruction. An interrupt may be generated by a number of sources, which may be either internal or external to the CPU. There are basically two types of interrupts : **hardware interrupts and software interrupts.**

Hardware interrupts are used by devices to communicate that they require attention from the operating system. Internally, hardware interrupts are implemented using electronic alerting signals that are sent to the processor from an external device, which is either a part of the computer itself, such as a disk controller, or an external peripheral. For example, pressing a key on the keyboard or moving the mouse triggers hardware interrupts that cause the processor to read the keystroke or mouse position. Unlike the software type (described below), hardware interrupts are asynchronous and can occur in the middle of instruction execution, requiring additional care in programming. The act of initiating a hardware interrupt is referred to as an Interrupt Request (IRQ).

A software interrupt is caused either by an exceptional condition in the processor itself, or a special instruction in the instruction set which causes an interrupt when it is executed. The former is often called a **trap or exception** and is used for errors or events occurring during program execution that are exceptional enough that they cannot be handled within the program itself. Each interrupt has its own interrupt handler. The number of hardware interrupts is limited by the number of interrupt request (IRQ) lines to the processor, but there may be hundreds of different software interrupts. Interrupts are a commonly used technique for computer multitasking, especially in real-time computing. Such a system is said to be interrupt-driven.

Some of the basic issues of interrupt are :

- What are the different kinds of interrupt conditions ?
- What are the advantages of having an interruption mechanism ?
- How is the CPU informed about the occurrence of an interrupt ?
- What does the CPU do on occurrence of an interrupt ?

Table 1, gives the list of some common interrupts and events that cause the occurrence of those interrupts.

Interrupt Condition	Occurrence of Event
Interrupt are generated by executing program itself (also called traps)	* division by zero * The number exceeds the maximum allowed. * Attempt of executing an illegal/ privileged instruction * Trying to reference memory location other than allowed for that program.

Interrupt generated by clock in the processor	* Generally used on expiry of time allocated for a program, in multiprogramming operating systems.
Interrupts generated by I/O devices and their interfaces	* Request of starting an Input/Output operation. * Normal completion of an Input/Output operation. * Occurrence of an error in Input/Output operation.
Interrupts on Hardware failure	* Power failure * Memory parity error.

Table 1 : Types of Interrupts and their causes

Interrupts are a useful mechanism. They are useful in improving the efficiency of processing. **How ?** This is to the fact that almost all the external devices are slower than the processor, therefore, in a typical system, a processor has to continually test whether an input value has arrived or a printout has been completed, in turn wasting a lot of CPU time. With the interrupt facility CPU is freed from the task of testing status of Input/Output devices and can do useful processing during this time, thus increasing the processing efficiency. Now the most important question is : How does the CPU know that an interrupt has occurred ?

There needs to be a line or a register or status word in CPU that can be raised on occurrence of interrupt condition. Once a CPU knows that an interrupt has occurred then following need to be done :

- First the condition is to be checked as to why the interrupt has occurred. That includes not only the device but also why that device has raised the interrupt.
- Once the interrupt condition is determined the necessary program called **ISRs (Interrupt Servicing Routines)** must be executed such that the CPU can resume further operations.

For example, assume that the interrupt occurs due to an attempt by an executing program for execution of an illegal or privileged instruction, then ISR for such interrupt may terminate the execution of the program that has caused this condition. Thus, on occurrence of an Interrupt the related ISR is executed by the CPU. The ISRs are pre-defined programs written for specific interrupt conditions.

Considering these requirements let us work out the steps, which CPU must perform on the occurrence of an interrupt.

- The CPU must find out the source of the interrupt, as this will determine which interrupt service routine is to be executed.
- The CPU then acquires the address of the interrupt service routine, which are stored in the memory (in general).

- What happens to the program the CPU was executing before the interrupt ? this program needs to be interrupted till the CPU executes the Interrupt Service Program. Do we need to do something for this program ? Well the context of this program is to be saved. We will discuss this a bit later.
- Finally, the CPU executes the interrupt service routine till the completion of the routine. A **RETURN** statement marks the end of this routine. After that, the control is passed back to the interrupted program.

Let us first discuss saving the context of a program. The execution of a program in the CPU is done using certain set of registers and their respective circuitry. As the CPU register are also used for execution of the interrupt service routine, it is highly likely that these routines alter the content of several registers. Therefore, it is the responsibility of the operating system that before an interrupt service routine is executed the previous content of the CPU registers should be stored, such that the execution of interrupted program can be restarted without any change from the point of interruption. Therefore, at the beginning of interrupt processing the essential context of the processor is saved either into a special save area in main memory or into a stack. This context is restored when the interrupt service routine is finished, thus, the interrupted program execution can be restarted from the point of interruption.

1.4.1 Interrupts and Instruction Cycle

Let us summaries the interrupt process, on the occurrence of an interrupt, an interrupt request (in the form of a signal) is issued to the CPU. The CPU on receipt of interrupt request suspends the operation of the currently executing program, saves the context of the currently executing program and starts executing the program which service that interrupt request. This program is also known as **interrupt handler.** After the interrupting condition/device has been serviced the execution of original program is resumed.

Thus, an interrupt can be considered as the interruption of the execution of an ongoing user program. The execution of user program resumes as soon as the interrupt processing is completed. Therefore, the user program does not contain any code for interrupt handling. This job is to be performed by the processor and the operating system, which in turn are also responsible for suspending the execution of the user program, and later after interrupt handling, resumes the user program from the point of interruption.

But when can a user program execution be interrupted ?

It will not be desirable to interrupt a program while an instruction is getting executed and is in a state like instruction decode. The most desirable place for program interruption would be when it has completed the previous instruction and is

about to start a new instruction. The complete interrupt cycle is given below in the **Figure 4.**

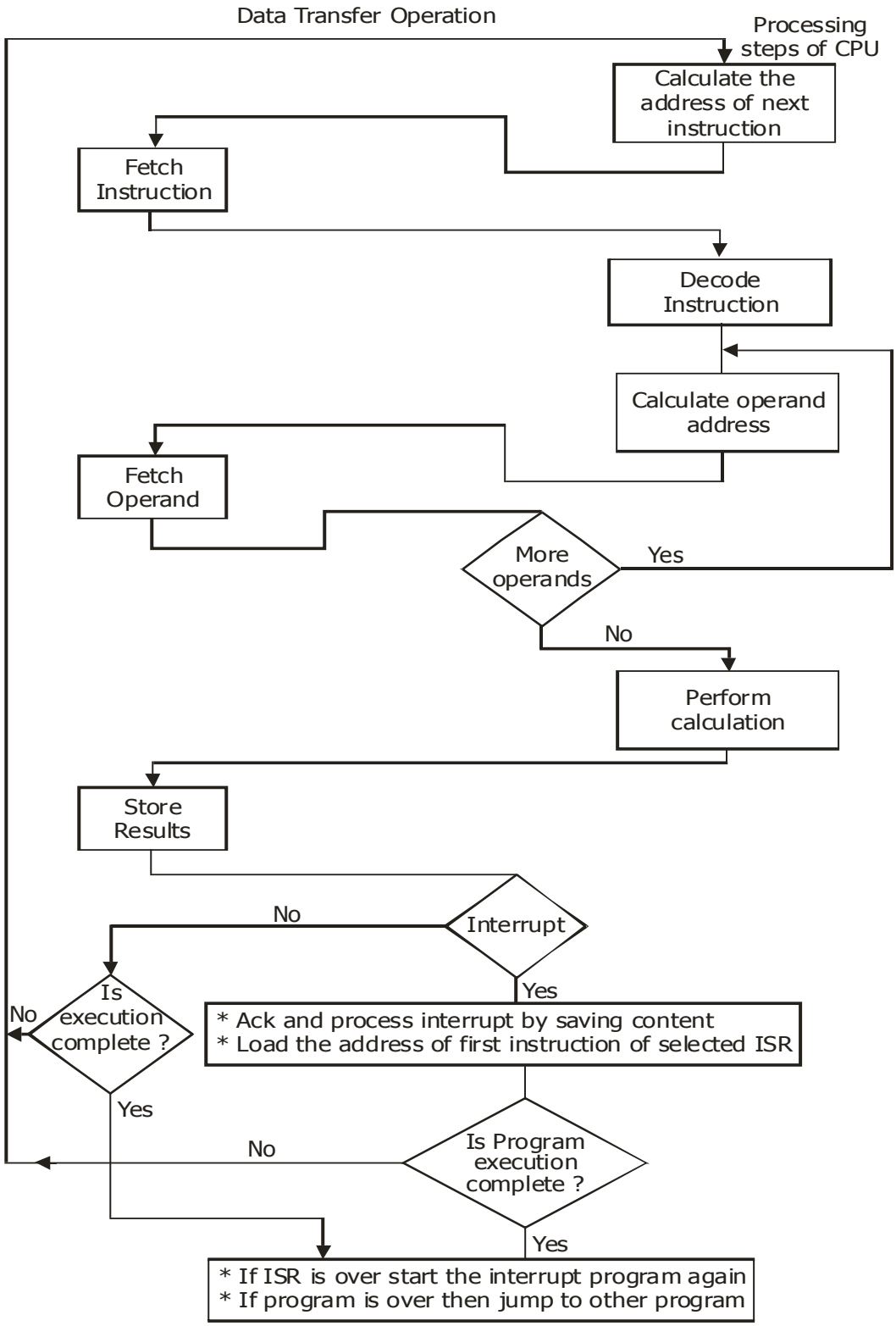


Figure 4 : Instruction Cycle with Interrupt Cycle

Figure 4. shows instruction execution cycle with interrupt cycle, where the interrupt condition is acknowledged. Please note that even interrupt service routine is also a program and after acknowledging interrupt the next instruction executed through instruction cycle is the first instruction of interrupt servicing routine.

In the interrupt cycle, the responsibility of the CPU/Processor is to check whether any interrupts have occurred checking the presence of the interrupt signal. In case no interrupt needs service, the processor proceeds to the next instruction of the current program. In case an interrupt needs servicing then the interrupt is processed as per the following.

- Suspend the execution of current program and save its context.
- Set the Program counter to the starting address of the interrupt service routine of the interrupt acknowledged.
- The processor then executes the instructions in the interrupt-servicing program.
- After completing the interrupt servicing program the CPU can resume the program it has suspended in step 1 above.

1.5 Data Representation

A computer must be able to take input, process it and produce output. The two most important questions and their respective answers here are :

Q.1. How is the Information represented in a computer ?

Ans. It is in the form of **B**inary Digi**t** popularly called **Bit**.

Q.2. How is the input and output presented in a form that is understood by us ?

Ans. One of the minimum requirements in this case may be to have a representation for characters. Thus, a mechanism that fulfils such requirement is needed. In Computers information is represented in digital form, therefore, to represent characters in computer we need codes. Some common character codes are **ASCII, EBCDIC** etc. These character codes are discussed in the sections below.

1.6 Number System

Number system is used to represent information in quantitative form. Some of the common number system are **binary, octal, decimal and hexadecimal**.

A number system of base (also called radix) **r** is a system which has r distinct symbols for r digits. A string of these symbolic digits represents a number. To determine the value that a number represents, we multiply the number by its place value that is an integer power of **r** depending on the place it is located and then find the sum of weighted digits. (We will use subscript **8** for an octal number, similarly, a subscript **2** for binary, **10** for decimal and **16** for hexadecimal number. In case no subscript is specified then number will be treasted as decimal number of else whatever number system is specified before it .)

Decimal Numbers : Decimal number system has ten digits represented from **0 to 9**. Any decimal number can be represented as a string of these digits and since there are ten decimal digits, therefore, the base or radix of this system is **10.**

Thus, a number 60.5 can be represented as :

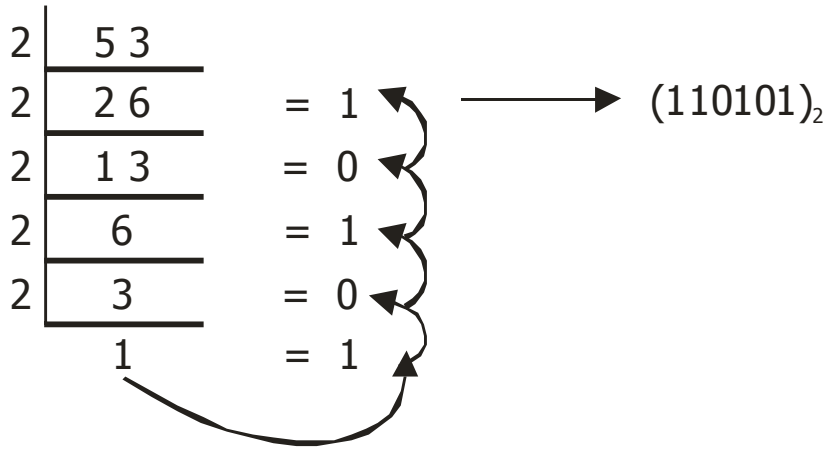
$$\mathbf{6 \times 10^1 + 0 \times 10^0 + 5 \times 10^{-1}}$$

Binary numbers : In binary numbers we have two digits 0 and 1 and they can be represented, as a string of these two-digits called bits. The base of binary number system is 2. A group of **4** bits and **8** bits called a **nibble** and **byte** respectively. The highest decimal number that can be represented by n bits binary number is $\mathbf{2^n - 1}$. Thus with an 8 bit number the maximum decimal number that be represented is $\mathbf{2^8 - 1}$ is 255.

For example 53.625 is a decimal number which can be **converted** to is binary equivalent as follows:

Step 1 : The integer is 53 and fraction is 0.625.

Step 2 : Integer conversion.



We read the remainders from bottom to top which give the binary equivalent.

Therefore, $(53)_{10} = (110101)_2$

Step 3 : Fractional conversion : If the decimal number is a fraction, its binary equivalent is obtained by multiplying the number continuously by 2, recording a carry in the integer position each time. The carrier in the forward order give the required binary number

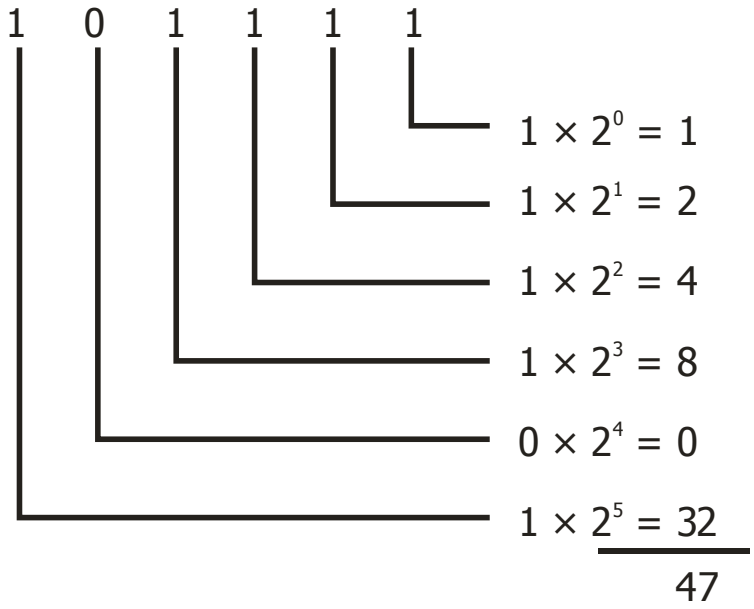
Multiplication		**Generated Integer**
0.625×2	=	$1.25 \rightarrow 1$
0.250×2	=	$0.50 \rightarrow 0$
0.500×2	=	$1.00 \rightarrow 1$
0.000×2	=	$0.00 \rightarrow 0$

Further multiplication by two is not possible since the product is zero. The binary equivalent is obtained by reading the carry terms from top to bottom. The combined number will give the binary equivalent as :

$$(53.625)_{10} = (110101.101)_2$$

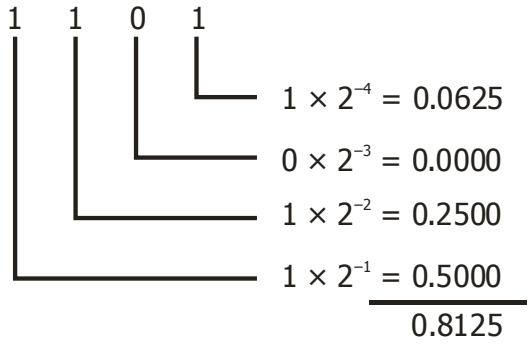
Binary to Decimal Conversion :

For example 101111.1101 is a valid Binary number. A binary number can be converted into a decimal number by multiplying the Binary numbers 1 or 0 by their weights and adding the products.



Therefore $(101111)_2 = (47)_{10}$

Conversion of $(.1101)_2$ is done is a similar manner.



Therefore $(0.1101)_2 = (0.8125)_{10}$

Combining the numbers we have

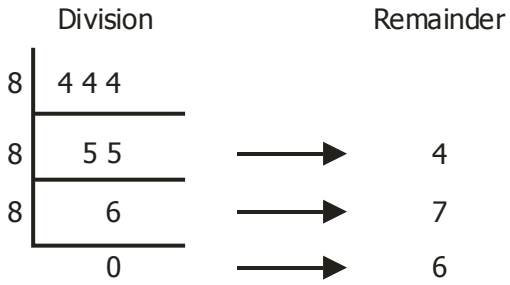
$$(101111.1101)_2 = (47.8125)_{10}$$

Octal Numbers : An Octal number use the digits from 0 to 7. The base or radix of octal number is 8. Each significant position in an octal number has a positional weight. The least significant position has a weight of 8^0 that is 1.

The octal equivalent of a decimal number can be obtained by dividing given decimal number by 8 repeatedly until quotient of 0 is obtained. We will explain the decimal to octal conversion in the following example :

Ex : Convert $(444.456)_{10}$ to an octal number.

Step 1 : Integer conversion :



Reading the remainders from bottom to top, the decimal number $(444)_{10} = (674)_8$.

Step 2 : Fractional conversion :

Multiplication	**Generated Integer**
$0.456 \times 8 = 3.648 \rightarrow$	3
$0.648 \times 8 = 5.184 \rightarrow$	5
$0.184 \times 8 = 1.472 \rightarrow$	1
$0.472 \times 8 = 3.776 \rightarrow$	3
$0.776 \times 8 = 6.208 \rightarrow$	6

The process is terminated then significant digits are obtained. Thus combining the integer and fractional part we have

$$(444.456)_{10} = (674.35136)_8$$

Octal to decimal conversion

Ex. (i) convert $(120)_8$ to its decimal equivalent.

Solv $\rightarrow (120)_8 = 1 \times 8^2 + 2 \times 8^1 + 0 \times 8^0$

$= 1 \times 64 + 2 \times 8 + 0$

$= 64 + 16 + 0$

$= (80)_{10}$.

Octal to Binary conversion : Conversion from octal to Binary and vice-versa can be easily carried out. For obtaining the Binary equivalent of an octal number, each significant digit in the given number is replaced by its 3-bit binary equivalent. For example :

$(376)_8 =$ 3 7 6

$=$ 011 111 110 $\rightarrow$ binary equivalent

Thus $(376)_8 = (011\ 111110)_2$.

Similarly for converting binary to its octal equivalent, we reverse the process, as follows

$$(1\ \overbrace{010}\ \overbrace{111}\ \overbrace{111}\ \overbrace{101})_2 = 001\ \ 010\ \ 111\ \ 111\ \ 101$$

$$1\ \ 2\ \ 7\ \ 7\ \ 5$$

Thus $(1010111111101)_2 = (12\ 775)_8$

(* we make a group of 3-bits from left to right).

Hexadecimal Number : The hexadecimal number system has base or radix of 16. It use symbols, namely 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. the symbols A to F represent the decimal 10, 11, 12, 13, 14 and 15 respectively. Each significant position in an hexadecimal number has a positional weight. The hexadecimal equivalent of a decimal number can be obtained by dividing the given decimal number by 16 respectively until a quotient of 0 is obtained.

Ex. Convert $(235)_{10}$ to hexadecimal number

Division			Remainder
16	2 3 5		
16	1 4	11 →	B
	0	14 →	E

Reading the remainders from bottom to top, the decimal number $(235)_{10}$ = $(EB)_{16}$.

The conversion from hexadecimal to its decimal number can be carried out by multiplying each significant digit of hexadecimal by its respective weight and adding the product. For example.

Convert $(A\ 3\ B)_{16}$ to its decimal equivalent.

Solv. $(A\ 3\ B)_{16} = A \times 16^2 + 3 \times 16^1 + B \times 16^0$

$= 10 \times 16^2 + 3 \times 16^1 + 11 \times 1$

$= 10 \times 256 + 3 \times 16 + 11$

$= 2560 + 48 + 11$

$= (2619)_{10}$

Hexadecimal to Binary conversion

Conversion from hexadecimal to binary and vice-versa can easily be carried out by putting four bit binary equivalent for each hexadecimal number.

For example :

$(2\ D\ 5)_{16}$ = 2 D 5

0010 1101 0101

Thus $(2\ D\ 5)_{16} = (001011010101)_2$.

The reverse procedure is used for converting a binary number to an hexadecimal. For example.

$(\overline{111}\ \overline{1011}\ \overline{0101})_2 = 0111\ \ 1011\ \ 0101$

7 B 5

Thus $(11110110101)_2 = (7\ B\ 5)_{16}$

Hexadecimal Octal Conversion

Conversion from hexadecimal to octal and vice-versa is sometimes required. To convert a hexadecimal number to octal, the following steps are applied :

(i) Convert the given hexadecimal number to its binary equivalent using 4 bits.

(ii) For m group of 3 bits starting from least significant bit (LSB).

(iii) Write the equivalent octal number for each group of 3 bits.

For example,

$$(47)_{16} = (\overbrace{01}^{\text{MSB}}\overbrace{00\quad 0}\ \overbrace{111}^{\text{LSB}})_2$$
$$= \ 1\ \ 0\qquad 7 = (107)_8$$

* (Try and convert octal to hexadecimal.)

Floating point representation of number

In the decimal system very large and very small numbers are expressed in scientific notation as follows : 4.69×10^{23} and 1.601×10^{-19}. Binary number can also be expressed by the floating point representation. The floating point representation of a number consist of the parts :

(i) The first part represent a signed, fixed point number called the mantissa (m);

(ii) Second part designates the positon of the decimal (or Binary) point and called the exponent (e).

The fixed point mantissa may be fraction or an integer. The number of bits required to express the exponent and mantissa is determined by the accuracy desired from the computing system as well as its capability to handle such numbers. For example, the decimal number + 8145.679 is represented in floating point as follows :

$$\underbrace{\overset{\text{Sign}}{0}\qquad 0.8145679}_{\text{Mantissa}} \qquad \underbrace{\overset{\text{Sign}}{0}\qquad 04}_{\text{Exponent}}$$

The mantissa has a 0 in the leftmost position to denote a plus. Here, the mantissa is considered to be a fixed point fraction. This representation is equivalent to the number expressed as a fraction 10 times by an exponent, that is $0.8145679 \times 10^{+4}$. Mantissa is sometimes called the fraction part.

Let us consider for example a computer that assumes integer representation for the mantissa and radix 8 for the numbers. The octal number $+36.754 = 36754 \times 8^{-3}$ in its floating point representation will look like this :

$$\underbrace{\overset{\text{Sign}}{0} \quad 36754}_{\text{Mantissa}} \qquad \underbrace{\overset{\text{Sign}}{1} \quad 03}_{\text{Exponent}}$$

When this number is represented in a register in its binary-coded form, the actual value of the register becomes :

$$\frac{0}{0}\ \frac{0111}{3}\ \frac{110}{6}\ \frac{111}{7}\ \frac{101}{5}\ \frac{100}{4} \text{ and } \frac{1}{1}\ \frac{000}{0}\ \frac{011}{3}$$

Most computer and all electronic calculators have a built in capacity to perform floating point arithmetic operations.

Institute of Electrical and Electronics Engineers (IEEE) is a society which has created lots of standards regarding various aspects of computer, has created IEEE standard 754 for floating point representation and arithmetic. The basic objective of developing this standard has to facilitate the portability of program from one to another computer. This standard has resulted in development of standard numerical capabilities in various microprocessors.

1.7 Decimal Representation in Computers

Normally, the decimal digits are coded in 7 ro 8 bits as alphanumeric characters. But for the purpose of arithmetic calculation the decimal digits are treated as four bit binary code. As we know that 2 binary bits can represent $2^2 = 4$ different combinations, 3-bits can represent $2^3 = 8$ combination and 4-bit can represent $2^4 = 16$ combinations. To represent decimal digits into binary form we use 4-bit for each decimal digits from 0 to 9. This is called **Binary-coded-Decimal (BCD)** number. The table below shows the BCD numbers.

Decimal Number	Binary Number	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001

Decimal Number	Binary Number	BCD
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

Let us represent 86.065 in BCD

8	6	•	0	6	5
1000	0110	•	0000	0110	0101

* If we try to compare the equivalent BCD with equivalent Binary value, we will find that both are different.

1.8 Alphanumeric Representation

The codes that represent numbers, alphabetic letters and special symbols are called alphanumeric codes. A complete set of necessary characters include :

(i) 26 lower case letters

(ii) 26 upper case letters

(iii) 10 numeric digits (0 to 9).

(iv) about 25 special characters such as 4, /, #, $ ….. etc.

These total upto about 87 symbols. To represent these 87 symbols with some binary code will require at least 7-bits.

ASCII Code

The full form of ASCII is American Standard Code for Information Interchange. It uses 7 bits to represent 2^7 = 128 characters. ASCII was extended to 8 bits to represent 2^8 = 256 characters called extended ASCII codes. The extended ASCII codes are the codes used in most of the microcomputers.

EBCDIC

Another alphanumeric code used in IBM equipment is the EBCDIC or extended Binary Coded Decimal Information code. It differs from ASCII only in its code grouping for the different alphanumeric characters. It uses eight bits for each character.

ISCII (Indian Standard Code for Information Interchange).

It is an eight-bit code that contains the standard ASCII values till 127 from 128-225. It contains the characters required in the ten Brahmi-based. Indian scripts. It is defined in IS 13194 : 1991 standard. It support INSCRIPT keyboard which provides a logical arrangement of vowels and consonants based on the phonetic properties and usage frequencies of the letter of Brahmi-Scripts. Any software that uses ISCII codes can be used in any Indian script, enhancing its commercial viability.

UNICODE

This is a newer International standard for character representation. Unicode provides a unique number for every character, no matter what the platform, no matter what the program and no matter what the language is. The Unicode standard has been adopted by industry leaders like Apple, HP, IBM, Microsoft, ORACLE, SAP etc.

One of the advantage of Unicode in the client server or multi-tiered applications and websites is the cost saving over the use of legacy character sets that results in targeting websites and software products across multiple plateform, languages and countries without re-engineering. It allows dots to be transported through many different systems without corruption.

1.9 Error Dectection and Correction Codes

Before ending the unit we must discuss about the errors, how it is detected and corrected.

During the process of binary dots transmission, errors may occur. In order to detect and correct such errors, two types of codes, namely (i) error-detecting codes and (ii) error-correcting codes, may be used.

If a single error transform a valid code word into an invalid one, it is said to be single error detecting code. The most simple and commonly used error detecting method is the parity check. In this method an extra parity bit is included with the binary message to make the total number of $1's$ either odd or even. This results in either even parity or odd parity. In even parity method the total number of $1's$ in the code group (including the parity bit) must be an even number. Similarly, in the odd-parity method, the total number of $1's$ (including the parity bit) must be an odd number. The parity bit can be placed at the either end of the code. Table 2 shows a message of 4 bits and its corresponding even and odd parity bits.

Message	Even-parity code (P)		Odd-parity-code (I)	
		P		(P)
0000	0000	0	0000	1
0001	0001	1	0001	0
0101	0101	0	0101	1
1110	1110	1	1110	0

If a single error occurs, it transform the valid code into an invalid one. This helps in the detection of single bit errors. Though the parity code is meant for single error detection, it can detect only odd number of errors. However in both the cases, original code word can not be found. If an even number of error occurs, then the parity check is satisfied, giving erroneous results.

Hamming : Error detection and correction codes

R. W Hamming developed a system that provides a methodical way to add one or more parity bits to a data character in order to detect and correct errors. The code words is defined as the number of bits changed from one code word to another.

Let us assume Ci and Cj be any two words in a particular block code. Harming distance dij between the two words Ci and Cj is defined by the number of bits in which they differ. The minimum value of dij is called Hamming distance **dmin**. For linear block codes minimum weight is equal to minimum distance.

For example

Ci = 1 0 0 0 1 1 1

↓ ↓ ↓

Cj = 0 0 0 1 0 1 1

Here, these codes differ in the left most bit position and in the 4th and 5th position from the left. Therefore dij = 3. From Humming's analysis of code distances the following important properties have been derived :

(i) A minimum distance of at least two is required for single error deletion.

(ii) Since the number of errors, $E \leq \frac{[d\min - 1]}{2}$ a minimum distance of three is required for single error correction.

(iii) Greater distance will provide deletion and/or correction of more number of errors.

The 7 bit Hamming (7,4) code word $h_1\ h_2\ h_3\ h_4\ h_5\ h_6\ h_7$ associated with a form bit binary number $b_3\ b_2\ b_1\ b_0$ is :

$h_1 = b_3 \oplus b_2 \oplus b_0$ $\quad$ $h_3 = b_3$

$h_2 = b_3 \oplus b_1 \oplus b_0$ $\quad$ $h_5 = b_2$

$h_4 = b_2 \oplus b_1 \oplus b_0$ $\quad$ $h_6 = b_1$

$h_7 = b_0$

Where $\oplus$ denoted the exclusive OR operation. Note that bits h1, h2 and h4 are even parity bits for the bit fields $b_3\ b_2\ b_0$, $b_3\ b_1\ b_0$ and $b_2\ b_1\ b_0$, respectively. In general, the parity bits (h_1, h_2, h_4, h_8 ……) are located in the positions corresponding to ascending power of two (i.e, 2^0, 2^1, 2^2, 2^3 ….. = 1, 2, 4, 8 …..)

To decode a Hamming code, one must check for odd parity over the bit fields in which even parity has already been taken out. For example, a single bit error is indicated by a non – zero parity word $c_4\ c_2\ c_1$, where

$c_1 = h_1 \oplus h_3 \oplus h_5 \oplus h_7$

$c_2 = h_2 \oplus h_3 \oplus h_6 \oplus h_7$

$c_4 = h_4 \oplus h_5 \oplus h_6 \oplus h_7$

If $C_4\ C_2\ C_1 = 0\ 0\ 0$, there is no error in the Hamming code. If it has non zero value, it indicates the bit position in error. For example, if $C_4\ C_2\ C_1 = 101$, then bit 5 is in error. To correct this error, bit 5 has to be complemented.

Example 1: Encode data bits 0 1 0 1 into a 7 bit even parity Hamming code.

Solution: Given $b_3\ b_2\ b_1\ b_0 = 0\ 1\ 0\ 1$.

Therefore,

$h_1 = b_3 \oplus b_2 \oplus b_0 = 0 \oplus 1 \oplus 1 = 0$

$h_2 = b_3 \oplus b_1 \oplus b_0 = 0 \oplus 0 \oplus 1 = 1$

$h_4 = b_2 \oplus b_1 \oplus b_0 = 1 \oplus 0 \oplus 1 = 0$

$h_3 = b_3 = 0$

$h_5 = b_2 = 1$

$h_6 = b_1 = 0$

$h_7 = b_0 = 1$

Therefore, $h_1\ h_2\ h_3\ h_4\ h_5\ h_6\ h_7$

0 1 0 0 1 0 1

Example 2 : A 7 bit Hamming code is received as 0 1 0 11 0 1. What is its correct code ?

Solution : h1 h2 h3 h4 h5 h6 h7

0 1 0 1 1 0 1

Now, to find the errors

$C_1 = h_1 \oplus h_3 \oplus h_5 \oplus h_7 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$

$C_2 = h_2 \oplus h_3 \oplus h_6 \oplus h_7 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$

$C_4 = h_4 \oplus h_5 \oplus h_6 \oplus h_7 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$

Thus , $C_4\ C_2\ C_1$ = 100. Therefore, bit 4 is in error and corrected code word can be obtained by complementing the fourth bit in the received code word as 0 1 0 0 1 0 1.

1.10 Summary

In this unit we completed the basic concepts of computer architecture. We discussed the von-Neumann architecture. This unit provides an in-depth coverage of instruction cycle, interrupts, and the data representation in a computer system. We have also covered aspects relating to error detection mechanism. The unit covers number system, conversion of number system, conversion of number to a different number system. It introduces the concept of computer arithmetic using 2's complement notation and provides introduction to information representation codes like ASCII EBCDIC, etc. The concept of floating point numbers has also been covered. Finally error detection and correction mechanism is detailed. The information given on various topics such as data representation, error detection codes etc, is although exhaustive yet can be supplemented with additional reading

In fact, a course in an area of computer must be supplemented by further reading to keep your knowledge up to date, as the computer world is changing with leaps and bounds. In addition to further readings the student is advised to study several Indian journals on computers to enhance his knowledge.

1.11 Questions

1. What is von Neumann Architecture ? Describe the block diagram of von Neumann.
2. What is an instruction cycle ? Explain each step of instruction cycle.
3. What is the role of memory in a computer ?
4. Define interrupt. Describe different types of interrupt condition and the events that causes the occurrence of those interrupts.
5. Explain the flowchart of instruction cycle with interrupt cycle.
6. Convert the following binary numbers to decimal.

   (i) 11001 (ii) 1010.10

   (iii) 11101.110 (iv) 010101.001

7. Convert the following decimal numbers to binary, octal and hexadecimal.

   (i) 65 (ii) 861 (iii) 1234 (iv) 2656.01

   (v) 6060.60 (vi) 3426.006

8. Convert the following binary numbers to octal and hexadecimal.

   (i) 11010101 (ii) 1110110.001

   (iii) 1111010.010 (iv) 100011.0100

9. Write the BCD equivalent of the given numbers.

   (i) 45 (ii) 73 (iii) 146 (iv) 621.10

   (v) 553.02 (vi) 653.321

10. Find the even and odd parity bits for the following 7-bit dots.

    (i) 0101010 (ii) 0001011 (iii) 1110101 (iv) 1001001

11. Explain Hamming error correcting codes.

12. A 7-bit Hamming code is received as 0110010. What is its correct code.

13. Encode data bits 1101 into a 7-bit even parity Hamming code.

1.12 Suggested Readings

1. Digital Logic and computer Design by Morris Mono.

2. Computer Organisation and Architecture, William Stallings.

●●●