

Module 1 : Introduction

In this Module, we have three lectures, viz.

1. [Introduction to computer System and its submodules](#)
2. [Number System and Representation of information.](#)
3. [Brief History of Comp. Evolution](#)

Click the proper link on the left side for the lectures

Representation of Basic Information

The basic functional units of computer are made of electronics circuit and it works with electrical signal. We provide input to the computer in form of electrical signal and get the output in form of electrical signal.

There are two basic types of electrical signals, namely, **analog** and **digital**. The analog signals are continuous in nature and digital signals are discrete in nature.

The electronic device that works with continuous signals is known as **analog device** and the electronic device that works with discrete signals is known as **digital device**. In present days most of the computers are digital in nature and we will deal with Digital Computer in this course.

Computer is a digital device, which works on two levels of signal. We say these two levels of signal as **High** and **Low**. The High-level signal basically corresponds to some high-level signal (say 5 Volt or 12 Volt) and Low-level signal basically corresponds to Low-level signal (say 0 Volt). This is one convention, which is known as positive logic. There are others convention also like negative logic.

Since Computer is a digital electronic device, we have to deal with two kinds of electrical signals. But while designing a new computer system or understanding the working principle of computer, it is always difficult to write or work with 0V or 5V. To make it convenient for understanding, we use some logical value, say,

LOW (L) - will represent 0V and

HIGH (H) - will represent 5V

Computer is used to solve mainly numerical problems. Again it is not convenient to work with symbolic representation. For that purpose we move to numeric representation. In this convention, we use 0 to represent **LOW** and 1 to represent **HIGH**.

0	means	LOW
1	means	HIGH

To know about the working principle of computer, we use two numeric symbols only namely 0 and 1. All the functionalities of computer can be captured with 0 and 1 and its theoretical background corresponds to two valued boolean algebra.

With the symbol 0 and 1, we have a mathematical system, which is known as **binary number system**. Basically binary number system is used to represent the information and manipulation of information in computer. This information is basically strings of 0s and 1s.

The smallest unit of information that is represented in computer is known as Bit (Binary Digit), which is either 0 or 1. Four bits together is known as **Nibble**, and Eight bits together is known as **Byte**.

Introduction :: Computer Organization and Architecture

Computer technology has made incredible improvement in the past half century. In the early part of computer evolution, there were no stored-program computer, the computational power was less and on the top of it the size of the computer was a very huge one.

Today, a personal computer has more computational power, more main memory, more disk storage, smaller in size and it is available in affordable cost.

This rapid rate of improvement has come both from advances in the technology used to build computers and from innovation in computer design. In this course we will mainly deal with the innovation in computer design.

The task that the computer designer handles is a complex one: Determine what attributes are important for a new machine, then design a machine to maximize performance while staying within cost constraints.

This task has many aspects, including instruction set design, functional organization, logic design, and implementation.

While looking for the task for computer design, both the terms computer organization and computer architecture come into picture.

It is difficult to give precise definition for the terms Computer Organization and Computer Architecture. But while describing computer system, we come across these terms, and in literature, computer scientists try to make a distinction between these two terms.

Computer architecture refers to those parameters of a computer system that are visible to a programmer or those parameters that have a direct impact on the logical execution of a program. Examples of architectural attributes include the instruction set, the number of bits used to represent different data types, I/O mechanisms, and techniques for addressing memory.

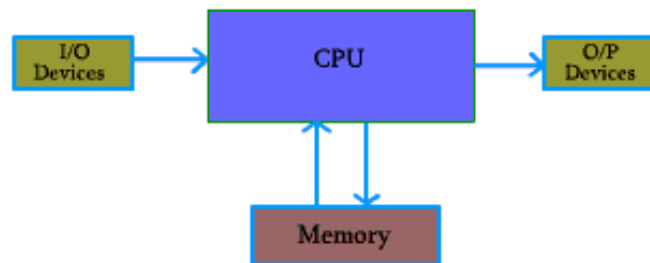
Computer organization refers to the operational units and their interconnections that realize the architectural specifications. Examples of organizational attributes include those hardware details transparent to the programmer, such as control signals, interfaces between the computer and peripherals, and the memory technology used.

In this course we will touch upon all those factors and finally come up with the concept how these attributes contribute to build a complete computer system.

Basic Computer Model and different units of Computer

The model of a computer can be described by four basic units in high level abstraction. These basic units are:

- Central Processor Unit
- Input Unit
- Output Unit
- Memory Unit



Basic Computer Model and different units of Computer

A. Central Processor Unit [CPU] :

Central processor unit consists of two basic blocks :

- The program control unit has a set of registers and control circuit to generate control signals.
- The execution unit or data processing unit contains a set of registers for storing data and an Arithmetic and Logic Unit (ALU) for execution of arithmetic and logical operations.

In addition, CPU may have some additional registers for temporary storage of data.

B. Input Unit :

With the help of input unit data from outside can be supplied to the computer. Program or data is read into main storage from input device or secondary storage under the control of CPU input instruction.

Example of input devices: Keyboard, Mouse, Hard disk, Floppy disk, CD-ROM drive etc.

C. Output Unit :

With the help of output unit computer results can be provided to the user or it can be stored in storage device permanently for future use. Output data from main storage go to output device under the control of CPU output instructions.

Example of output devices: Printer, Monitor, Plotter, Hard Disk, Floppy Disk etc.

D. Memory Unit :

Memory unit is used to store the data and program. CPU can work with the information stored in memory unit. This memory unit is termed as primary memory or main memory module. These are basically semi conductor memories.

There are two types of semiconductor memories -

- **Volatile Memory** : RAM (Random Access Memory).
- **Non-Volatile Memory** : ROM (Read only Memory), PROM (Programmable ROM)
EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM).

Secondary Memory :

There is another kind of storage device, apart from primary or main memory, which is known as secondary memory. **Secondary memories are non volatile memory** and it is used for permanent storage of data and program.

Example of secondary memories:

Hard Disk, Floppy Disk, Magenetic Tape	-----	These are magnetic devices,
CD-ROM	-----	is optical device
Thumb drive (or pen drive)	-----	is semiconductor memory.

Basic Working Principle of a Computer

Before going into the details of working principle of a computer, we will analyse how computer works with the help of a small hypothetical computer.

In this small computer, we do not consider about Input and Output unit. We will consider only CPU and memory module. Assume that somehow we have stored the program and data into main memory. We will see how CPU can perform the job depending on the program stored in main memory.

P.S. - Our assumption is that students understand common terms like program, CPU, memory etc. without knowing the exact details.

Consider the Arithmetic and Logic Unit (ALU) of Central Processing Unit :

Consider an ALU which can perform four arithmetic operations and four logical operations

To distinguish between arithmetic and logical operation, we may use a signal line,

- 0 - in that signal, represents an arithmetic operation and
- 1 - in that signal, represents a logical operation.

In the similar manner, we need another two signal lines to distinguish between four arithmetic operations.

The different operations and their binary code is as follows:

Arithmetic		Logical	
000	ADD	100	OR
001	SUB	101	AND
010	MULT	110	NAND
011	DIV	111	ADD

Consider the part of control unit, its task is to generate the appropriate signal at right moment.

There is an instruction decoder in CPU which decodes this information in such a way that computer can perform the desired task

The simple model for the decoder may be considered that there is three input lines to the decoder and correspondingly it generates eight output lines. Depending on input combination only one of the output signals will be generated and it is used to indicate the corresponding operation of ALU.

In our simple model, we use three storage units in CPU,

Two -- for storing the operand and

one -- for storing the results.

These storage units are known as register.

But in computer, we need more storage space for proper functioning of the Computer.

Some of them are inside CPU, which are known as register. Other bigger junk of storage space is known as primary memory or main memory. **The CPU can work with the information available in main memory only.**

To access the data from memory, we need two special registers one is known as **Memory Data Register (MDR)** and the second one is **Memory Address Register (MAR)**.

Data and program is stored in main memory. While executing a program, CPU brings instruction and data from main memory, performs the tasks as per the instruction fetch from the memory. After completion of operation, CPU stores the result back into the memory.

In next section, we discuss about memory organization for our small machine.

Main Memory Organization

Main memory unit is the storage unit, There are several location for storing information in the main memory module.

The capacity of a memory module is specified by the number of memory location and the information stored in each location.

A memory module of capacity 16×4 indicates that, there are 16 location in the memory module and in each location, we can store 4 bit of information.

We have to know how to indicate or point to a specific memory location. This is done by address of the memory location.

We need two operation to work with memory.

READ Operation: This operation is to retrieve the data from memory and bring it to CPU register

WRITE Operation: This operation is to store the data to a memory location from CPU register

We need some mechanism to distinguish these two operations READ and WRITE.

With the help of one signal line, we can differentiate these two operations. If the content of this signal line is 0, we say that we will do a READ operation; and if it is 1, then it is a WRITE operation.

To transfer the data from CPU to memory module and vice-versa, we need some connection. This is termed as **DATA BUS**.

The size of the data bus indicate how many bit we can transfer at a time. Size of data bus is mainly specified by the data storage capacity of each location of memory module.

We have to resolve the issues how to specify a particular memory location where we want to store our data or from where we want to retrieve the data.

This can be done by the memory address. Each location can be specified with the help of a binary address.

If we use 4 signal lines, we have 16 different combinations in these four lines, provided we use two signal values only (say 0 and 1).

To distinguish 16 location, we need four signal lines. These signal lines use to identify a memory location is termed as **ADDRESS BUS**. Size of address bus depends on the memory size. For a memory module of capacity of 2^n location, we need n address lines, that is, an address bus of size n.

We use a address decoder to decode the address that are present in address bus

As for example, consider a memory module of 16 location and each location can store 4 bit of information

The size of address bus is 4 bit and the size of the data bus is 4 bit

The size of address decoder is 4×16 .

There is a control signal named R/W.

If $R/W = 0$, we perform a READ operation and

if $R/W = 1$, we perform a WRITE operation

If the contents of address bus is 0101 and contents of data bus is 1100 and $R/W = 1$, then 1100 will be written in location 5.

If the contents of address bus is 1011 and $R/W=0$, then the contents of location 1011 will be placed in data bus.

In next section, we will explain how to perform memory access operation in our small hypothetical computer.

Memory Instruction

We need some more instruction to work with the computer. Apart from the instruction needed to perform task inside CPU, we need some more instructions for data transfer from main memory to CPU and vice versa.

In our hypothetical machine, we use three signal lines to identify a particular instruction. If we want to include more instruction, we need additional signal lines.

Instruction	Code	Meaning
1000	LDAI imm	Load register A with data that is given in the program
1001	LDAA addr	Load register A with data from memory location addr
1010	LDBI imm	Load register B with data
1011	LDBA addr	Load register B with data from memory location addr
1100	STC addr	Store the value of register C in memory location addr
1101	HALT	Stop the execution
1110	NOP	No operation
1111	NOP	No operation

With this additional signal line, we can go upto 16 instructions. When the signal of this new line is 0, it will indicate the ALU operation. For signal value equal to 1, it will indicate 8 new instructions. So, we can design 8 new memory access instructions.

We have added 6 new instructions. Still two codes are unused, which can be used for other purposes. We show it as **NOP** means No Operation.

We have seen that for ALU operation, instruction decoder generated the signal for appropriate ALU operation.

Apart from that we need many more signals for proper functioning of the computer. Therefore, we need a module, which is known as control unit, and it is a part of CPU. The control unit is responsible to generate the appropriate signal.

As for example, for LDAI instruction, control unit must generate a signal which enables the register A to store in data into register A.

One major task is to design the control unit to generate the appropriate signal at appropriate time for the proper functioning of the computer.

Consider a simple problem to add two numbers and store the result in memory, say we want to add 7 to 5.

To solve this problem in computer, we have to write a computer program. The program is machine specific, and it is related to the instruction set of the machine.

For our hypothetical machine, the program is as follows

Instruction	Binary	HEX	Memory Location
LDAI 5	1000 0101	8 5	(0, 1)
LDBI 7	1010 0111	A 7	(2, 3)
ADD	0000	0	(4)
STC 15	1100 1111	C F	(5, 6)
HALT	1101	D	(7)

[See the next Page for a flash Demo...](#)

Introduction to Computer System[Print this page](#)[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

Consider another example, say that the first number is stored in memory location 13 and the second data is stored in memory location 14. Write a program to Add the contents of memory location 13 and 14 and store the result in memory location 15.

Instruction	Binary	HEX	Memory Location
LDAA 13	1000 0101	8 5	(0, 1)
LDBA 14	1010 0111	A 7	(2, 3)
ADD	0000	0	(4)
STC 15	1100 1111	C F	(5, 6)
HALT	1101	D	(7)

One question still remain unanswerd: How to store the program or data to main memory. Once we put the program and data in main memory, then only CPU can execute the program. For that we need some more instructions.

We need some instructions to perform the input tasks. These instructions are responsible to provide the input data from input devices and store them in main memory. For example instructions are needed to take input from keyboard.

We need some other instructions to perform the output tasks. These instructions are responsible to provide the result to output devices. For example, instructions are needed to send the result to printer.

We have seen that number of instructions that can be provided in a computer depends on the signal lines that are used to provide the instruction, which is basically the size of the storage devices of the computer.

For uniformity, we use same size for all storage space, which are known as register. If we work with a 16-bit machine, total instructions that can be implemented is 2^{16} .

The model that we have described here is known as **Von Neumann Stored Program Concept**. First we have to store all the instruction of a program in main memory, and CPU can work with the contents that are stored in main memory. Instructions are executed one after another.

We have explained the concept of computer in very high level abstraction by omitting most of the details.

[As the course progresses we will explain the exact working principle of computer in more details.](#)

Main Memory Organization: Stored Program

The present day digital computers are based on stored-program concept introduced by Von Neumann. In this stored-program concept, programs and data are stored in separate storage unit called memories.

Central Processing Unit, the main component of computer can work with the information stored in storage unit only.

In 1946, Von Neumann and his colleagues began the design of a stored-program computer at the Institute for Advanced Studies in Princeton. This computer is referred as the IAS computer.

The structure of IAS computer is shown in the next page.

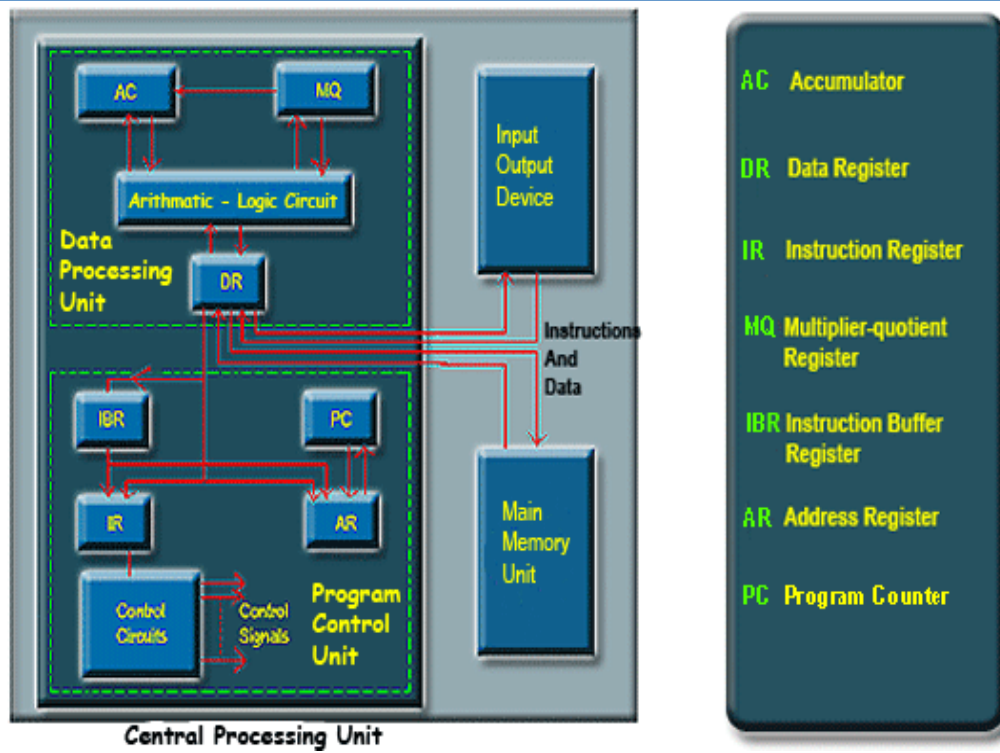


Figure : Structure of a first generation computer : IAS

The IAS computer is having three basic units:

- The Central Processing Unit (CPU).
- The Main Memory Unit.
- The Input/Output Device.

Central Processing Unit:

This is the main unit of computer, which is responsible to perform all the operations. The CPU of the IAS computer consists of a data processing unit and a program control unit.

The data processing unit contains a high speed registers intended for temporary storage of instructions, memory addresses and data. The main action specified by instructions are performed by the arithmetic-logic circuits of the data processing unit.

The control circuits in the program control unit are responsible for fetching instructions, decoding opcodes, controlling the information movements correctly through the system, and providing proper control signals for all CPU actions.

The Main Memory Unit:

It is used for storing programs and data. The memory locations of memory unit is uniquely specified by the memory address of the location. $M(X)$ is used to indicate the location of the memory unit M with address X .

The data transfer between memory unit and CPU takes place with the help of data register DR. When CPU wants to read some information from memory unit, the information first brings to DR, and after that it goes to appropriate position. Similarly, data to be stored to memory must put into DR first, and then it is stored to appropriate location in the memory unit.

The address of the memory location that is used during memory read and memory write operations are stored in the memory register AR.

The information fetched from the memory is a operand of an instruction, then it is moved from DR to data processing unit (either to AC or MQ). If it is an operand, then it is moved to program control unit (either to IR or IBR).

Two additional registers for the temporary storage of operands and results are included in data processing units: the accumulator AC and the multiplier-quotient register MQ.

Two instructions are fetch simultaneously from M and transferred to the program control unit. The instruction that is not to be executed immediately is placed in the instruction buffer register IBR. The opcode of the other instruction is placed in the instruction register IR where it is decoded.

In the decoding phase, the control circuits generate the required control signals to perform the specified operation in the instruction.

The program counter PC is used to store the address of the next instruction to be fetched from memory.

Input Output Device :

Input devices are used to put the information into computer. With the help of input devices we can store information in memory so that CPU can use it. Program or data is read into main memory from input device or secondary storage under the control of CPU input instruction.

Output devices are used to output the information from computer. If some results are evaluated by computer and it is stored in computer, then with the help of output devices, we can present it to the user. Output data from the main memory go to output device under the control of CPU output instruction.

Binary Number System

We have already mentioned that computer can handle with two type of signals, therefore, to represent any information in computer, we have to take help of these two signals.

These two signals corresponds to two levels of electrical signals, and symbolically we represent them as 0 and 1.

In our day to day activities for arithmetic, we use the **Decimal Number System**. The decimal number system is said to be of base, or radix 10, because it uses ten digits and the coefficients are multiplied by power of 10.

A decimal number such as 5273 represents a quantity equal to 5 thousands plus 2 hundres, plus 7 tens, plus 3 units. The thousands, hundreds, etc. are powers of 10 implied by the position of the coefficients. To be more precise, 5273 should be written as:

$$5 \times 10^3 + 2 \times 10^2 + 7 \times 10^1 + 3 \times 10^0$$

However, the convention is to write only the coefficient and from their position deduce the necessary power of 10.

In decimal number system, we need 10 different symbols. But in computer we have provision to represent only two symbols. So directly we can not use deciman number system in computer arithmetic.

For computer arithmetic we use **binary number system**. The binary number system uses two symbols to represent the number and these two symbols are 0 and 1.

The binary number system is said to be of base 2 or radix 2, because it uses two digits and the coefficients are multiplied by power of 2.

The binary number 110011 represents the quantity equal to:

$$1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 51 \text{ (in decimal)}$$

We can use binary number system for computer arithmetic.

Representation of Unsigned Integers

Any integer can be stored in computer in binary form. As for example:

The binary equivalent of integer 107 is 1101011, so 1101011 are stored to represent 107.

What is the size of Integer that can be stored in a Computer?

It depends on the word size of the Computer. If we are working with 8-bit computer, then we can use only 8 bits to represent the number. The eight bit computer means the storage organization for data is 8 bits.

In case of 8-bit numbers, the minimum number that can be stored in computer is 00000000 (0) and maximum number is 11111111 (255) (if we are working with natural numbers).

So, the domain of number is restricted by the storage capacity of the computer. Also it is related to number system; above range is for natural numbers.

In general, for n-bit number, the range for natural number is from 0 to $2^n - 1$

Any arithmetic operation can be performed with the help of binary number system. Consider the following two examples, where decimal and binary additions are shown side by side.

01101000	104
00110001	49
-----	-----
10011001	153

In the above example, the result is an 8-bit number, as it can be stored in the 8-bit computer, so we get the correct results.

10000001	129
10101010	178
-----	-----
100101011	307

In the above example, the result is a 9-bit number, but we can store only 8 bits, and the most significant bit (msb) cannot be stored.

The result of this addition will be stored as (00101011) which is 43 and it is not the desired result. Since we cannot store the complete result of an operation, and it is known as the overflow case.

The smallest unit of information is known as **BIT** (Binary digiT).

The binary number 110011 consists of 6 bits and it represents:

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

For an n-bit number the coefficient is - a_j multiplied by 2^j where, $(0 \leq j < n)$

The coefficient $a_{(n-1)}$ is multiplied by $2^{(n-1)}$ and it is known as most significant bit (MSB).

The coefficient a_0 is multiplied by 2^0 and it is known as least significant bit (LSB).

For our convenient, while writing in paper, we may take help of other number systems like octal and hexadecimal. It will reduce the burden of writing long strings of 0s and 1s.

Octal Number : The octal number system is said to be of base, or radix 8, because it uses 8 digits and the coefficients are multiplied by power of 8.

Eight digits used in octal system are: 0, 1, 2, 3, 4, 5, 6 and 7.

Hexadecimal number : The hexadecimal number system is said to be of base, or radix 16, because it uses 16 symbols and the coefficients are multiplied by power of 16.

Sixteen digits used in hexadecimal system are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

Consider the following addition example:

Binary	Octal	Hexadecimal	Decimal
01101000	150	68	104
00111010	072	3A	58
-----	-----	-----	-----
10100010	242	A2	162

Signed Integer

We know that for n-bit number, the range for natural number is from 0 to $2^n - 1$.

For n-bit, we have all together 2^n different combination, and we use these different combination to represent 2^n numbers, which ranges from 0 to $2^n - 1$.

If we want to include the negative number, naturally, the range will decrease. Half of the combinations are used for positive number and other half is used for negative number.

For n-bit representation, the range is from -2^{n-1} to $+2^{n-1} - 1$.

For example, if we consider 8-bit number, then range

for natural number is from 0 to 255; but
for signed integer the range is from -127 to +127.

Representation of signed integer

We know that for n-bit number, the range for natural number is from 0 to $2^n - 1$.

There are **three different schemes** to represent negative number:

- **Signed-Magnitude form.**
- **1's complement form.**
- **2's complement form.**

Signed magnitude form:

In signed-magnitude form, one particular bit is used to indicate the sign of the number, whether it is a positive number or a negative number. Other bits are used to represent the magnitude of the number.

For an n-bit number, one bit is used to indicate the signed information and remaining (n-1) bits are used to represent the magnitude. Therefore, the range is from $-2^{n-1} - 1$ to $+2^{n-1} - 1$.

Generally, Most Significant Bit (MSB) is used to indicate the sign and it is termed as signed bit. 0 in signed bit indicates positive number and 1 in signed bit indicates negative number.

Number System and Representation[Print this page](#)[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

For example, 01011001 represents **+** 169 and
 11011001 represents **-** 169

What is 00000000 and 10000000 in signed magnitude form?

The concept of complement

The concept of complements is used to represent signed number.

Consider a number system of base-r or radix-r. There are two types of complements,

- The radix complement or the r 's complement.
- The diminished radix complement or the $(r - 1)$'s complement.

Diminished Radix Complement :

Given a number N in base r having n digits, the $(r - 1)$'s complement of N is defined as $(r^n - 1) - N$.

For decimal numbers, $r = 10$ and $r - 1 = 9$, so the 9's complement of N is $(10^n - 1) - N$.

e.g., 9's complement of 5642 is $9999 - 5642 = 4357$.

[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

Radix Complement :

The r 's complement of an n -digit number in base r is defined as $(r^n - N)$ for $N \neq 0$ and 0 for $N = 0$.

r 's complement is obtained by adding 1 to the $(r - 1)$'s complement, since $(r^n - N) = [(r^n - 1) - N] + 1$

e.g., 10's complement of 5642 is 9's complement of 5642 + 1, i.e., 4357 + 1 = 4358

e.g., 2's complement of 1010 is 1's complement of 1010 + 1, i.e., 0101 + 1 = 0110.

Representation of Signed integer in 1's complement form:

Consider the eight bit number 01011100, 1's complements of this number is 10100011. If we perform the following addition:

If we add 1 to the number, the result is 100000000.

0	1	0	1	1	1	0	0
1	0	1	0	0	0	1	1

1	1	1	1	1	1	1	1

Since we are considering an eight bit number, so the 9th bit (MSB) of the result can not be stored. Therefore, the final result is 00000000.

Since the addition of two number is 0, so one can be treated as the negative of the other number. So, 1's complement can be used to represent negative number.

Representation of Signed integer in 2's complement form:

Consider the eight bit number 01011100, 2's complements of this number is 10100100. If we perform the following addition:

0	1	0	1	1	1	0	0
1	0	1	0	0	0	1	1
<hr/>							
1	0	0	0	0	0	0	0

Since we are considering an eight bit number, so the 9th bit (MSB) of the result can not be stored. Therefore, the final result is 00000000.

Since the addition of two number is 0, so one can be treated as the negative of the other number. So, 2's complement can be used to represent negative number.

Decimal	2's Complement	1's complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	-----	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	-----	-----

Representation of Real Number

Representation of Real Number:

Binary representation of 41.6875 is 101001.1011

Therefore any real number can be converted to binary number system

There are **two schemes** to represent real number :

- Fixed-point representation
- Floating-point representation

Fixed-point representation:

Binary representation of 41.6875 is 101001.1011

To store this number, we have to store **two information**,

- the part before decimal point and
- the part after decimal point.

This is known as fixed-point representation where the position of decimal point is fixed and number of bits before and after decimal point are also predefined.

If we use 16 bits before decimal point and 8 bits after decimal point, in signed magnitude form, the range is

$-2^{16} - 1$ to $+2^{16} - 1$ and the precision is $2^{(-8)}$

One bit is required for sign information, so the total size of the number is 25 bits

(1(sign) + 16(before decimal point) + 8(after decimal point)).

Floating-point representation:

In this representation, numbers are represented by a mantissa comprising the significant digits and an exponent part of Radix R. The format is:

$$\text{mantissa} * R^{\text{exponent}}$$

Numbers are often normalized, such that the decimal point is placed to the right of the **first non zero digit**.

For example, the decimal number,

$$5236 \text{ is equivalent to } 5.236 * 10^3$$

To store this number in floating point representation, we store 5236 in mantissa part and 3 in exponent part.

IEEE standard floating point format:

IEEE has proposed two standard for representing floating-point number:

- Single precision
- Double precision

Single Precision:

S: sign bit: 0 denoted + and 1 denotes -

E: 8-bit excess -27 exponent

M: 23-bit mantissa

Double Precision:

S: sign bit: 0 denoted + and 1 denotes -

E: 11-bit excess -1023 exponent

M: 52-bit mantissa

Representation of Character

Since we are working with 0's and 1's only, to represent character in computer we use strings of 0's and 1's only.

To represent character we are using some coding scheme, which is nothing but a mapping function.

Some of standard coding schemes are: **ASCII**, **EBCDIC**, **UNICODE**.

ASCII : American Standard Code for Information Interchange.

It uses a 7-bit code. All together we have 128 combinations of 7 bits and we can represent 128 character.

As for example 65 = 1000001 represents character 'A'.

EBCDIC : Extended Binary Coded Decimal Interchange Code.

It uses 8-bit code and we can represent 256 character.

UNICODE : It is used to capture most of the languages of the world. It uses 16-bit

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others.

A Brief History of Computer Architecture

Computer Architecture is the field of study of selecting and interconnecting hardware components to create computers that satisfy functional performance and cost goals. It refers to those attributes of the computer system that are visible to a programmer and have a direct effect on the execution of a program.

Computer Architecture concerns Machine Organization, interfaces, application, technology, measurement & simulation that Includes:

- **Instruction set**
- **Data formats**
- **Principle of Operation** (formal description of every operation)
- **Features** (organization of programmable storage, registers used, interrupts mechanism, etc.)

In short, it is the combination of Instruction Set Architecture, Machine Organization and the related hardware.

The Brief History of Computer Architecture

First Generation (1940-1950) :: Vacuum Tube

- **ENIAC [1945]:** Designed by Mauchly & Echert, built by US army to calculate trajectories for ballistic shells during World War II. Around 18000 vacuum tubes and 1500 relays were used to build ENIAC, and it was programmed by manually setting switches
- **UNIVAC [1950]:** the first commercial computer
- **John Von Neumann architecture:** Goldstine and Von Neumann took the idea of ENIAC and developed concept of storing a program in the memory. Known as the Von Neumann's architecture and has been the basis for virtually every machine designed since then.

Features:

- Electron emitting devices
- Data and programs are stored in a single read-write memory
- Memory contents are addressable by location, regardless of the content itself
- Machine language/Assembly language
- Sequential execution

Second Generation (1950-1964) :: Transistors

- William Shockley, John Bardeen, and Walter Brattain invent the **transistor** that reduce size of computers and improve reliability. Vacuum tubes have been replaced by transistors.
- **First operating Systems:** handled one program at a time
- **On-off switches** controlled by electronically.
- **High level languages**
- **Floating point arithmetic**

[Third Generation \(1964-1974\) :: Integrated Circuits \(IC\)](#)

- **Microprocessor chips** combines thousands of transistors, entire circuit on one computer chip.
- **Semiconductor memory**
- **Multiple computer models** with different performance characteristics
- The **size** of computers has been reduced drastically

[Fourth Generation \(1974-Present\) :: Very Large-Scale Integration \(VLSI\) / Ultra Large Scale Integration \(ULSI\)](#)

- **Combines** millions of transistors
- **Single-chip processor** and the single-board computer emerged
- Creation of the **Personal Computer (PC)**
- Use of **data communications**
- Massively **parallel machine**

Evolution of Instruction Sets

Instruction Set Architecture (ISA) Abstract interface between the Hardware and lowest-level Software

- 1950: **Single Accumulator**: EDSAC
- 1953: **Accumulator plus Index Registers**: Manchester Mark I, IBM 700 series
- **Separation of programming Model from implementation**:
 - 1963: High-level language Based: B5000
 - 1964: Concept of a Family: IBM 360
- **General Purpose Register Machines**:
 - 1977-1980: **CISC** - Complex Instruction Sets computer: Vax, Intel 432
 - 1963-1976: **Load/Store Architecture**: CDC 6600, Cray 1
 - 1987: **RISC**: Reduced Instruction Set Computer: Mips, Sparc, HP-PA, IBM RS6000

Typical RISC:

- Simple, no complex addressing
- Constant length instruction, 32-bit fixed format
- Large register file
- Hard wired control unit, no need for micro programming
- Just about every opposites of CISC

Major advances in computer architecture are typically associated with landmark instruction set designs. Computer architecture's definition itself has been through bit changes. The following are the main concern for computer architecture through different times:

- 1930-1950: Computer arithmetic
 - Microprogramming
 - Pipelining
 - Cache
 - Timeshared multiprocessor
- 1960: Operating system support, especially memory management
 - Virtual memory
- 1970-1980: **Instruction Set Design**, especially for compilers; **Vector processing** and **shared memory multiprocessors**
 - RISC
- 1990s: Design of CPU, memory system, I/O system, multi-processors, networks
 - CC-UMA multiprocessor
 - CC-NUMA multiprocessor
 - Not-CC-NUMA multiprocessor
 - Message-passing multiprocessor

- 2000s: Special purpose architecture, functionally reconfigurable, special considerations for low power/mobile processing, chip multiprocessors, memory systems
 - Massive SIMD
 - Parallel processing multiprocessor

Under a rapidly changing set of forces, computer technology keeps at dramatic change, for example:

- **Processor clock rate** at about 20% increase a year
- **Logic capacity** at about 30% increase a year
- **Memory speed** at about 10% increase a year
- **Memory capacity** at about 60% increase a year
- **Cost per bit** improves about 25% a year
- **The disk capacity** increase at 60% a year.

A Brief History of Computer Organization

If **computer architecture** is a view of the *whole design with the important characteristics* visible to programmer, **computer organization** is *how features are implemented with the specific building blocks* visible to designer, such as control signals, interfaces, memory technology, etc. Computer architecture and organization are closely related, though not exactly the same.

A stored program computer has the following basic units:

- **Processor** -- center for manipulation and control
- **Memory** -- storage for instructions and data for currently executing programs
- **I/O system** -- controller which communicate with "external" devices:
secondary memory, display devices, networks
- **Data-path & control** -- collection of parallel wires, transmits data, instructions, or control signal

Computer organization defines the ways in which these components are interconnected and controlled. It is the capabilities and performance characteristics of those principal functional units. Architecture can have a number of organizational implementations, and organization differs between different versions. Such, all **Intel x86** families share the same basic architecture, and **IBM system/370** family share their basic architecture.

The history of Computer Organization

Computer architecture has progressed five generation: **vacuum tubes**, **transistors**, **integrated circuits**, and **VLSI**. Computer organization has also made its historic progression accordingly.

The advance of microprocessor (Intel)

- **1977:** 8080 - the first general purpose microprocessor, 8 bit data path, used in first personal computer
- **1978:** 8086 - much more powerful with 16 bit, 1MB addressable, instruction cache, prefetch few instructions
- **1980:** 8087 - the floating point coprocessor is added
- **1982:** 80286 - 24 Mbyte addressable memory space, plus instructions
- **1985:** 80386 - 32 bit, new addressing modes and support for multitasking
- **1989 -- 1995:**
 - 80486 - 25, 33, MHz, 1.2 M transistors, 5 stage pipeline, sophisticated powerful cache and instruction pipelining, built in math co-processor.
 - Pentium - 60, 66 MHz, 3.1 M transistor, branch predictor, pipelined floating point, multiple instructions executed in parallel, first superscalar IA-32.
 - PentiumPro - Increased superscalar, register renaming, branch prediction, data flow analysis, and speculative execution
- **1995 -- 1997:** Pentium II - 233, 166, 300 MHz, 7.5 M transistors, first compaction of micro- architecture, MMX technology, graphics video and audio processing.
- **1999:** Pentium III - additional floating point instructions for 3D graphics
- **2000:** Pentium IV - Further floating point and multimedia enhancements

Evolution of Memory

- 1970: **RAM /DRAM**, 4.77 MHz
- 1987: **FPM** - fast page mode DRAM, 20 MHz
- 1995, **EDO** - Extended Data Output, which increases the read cycle between memory and CPU, 20 MHz
- 1997- 1998: **SDRAM** - Synchronous DRAM, which synchronizes itself with the CPU bus and runs at higher clock speeds, PC66 at 66 MHz, PC100 at 100 MHz
- 1999: **RDRAM** - Rambus DRAM, which DRAM with a very high bandwidth, 800 MHz
- 1999-2000: **SDRAM** - PC133 at 133 MHz, DDR at 266 MHz.
- 2001: **EDRAM** - Enhanced DRAM, which is dynamic or power-refreshed RAM, also know as cached DRAM.

Major buses and their features

A bus is a parallel circuit that connects the major components of a computer, allowing the transfer of electric impulses from one connected component to any other.

- **VESA** - Video Electronics Standard Association:
32 bit, relied on the 486 processor to function
- **ISA** - Industry Standard Architecture:
8 bit or 16 bit with width 8 or 16 bits. 8.3 MHz speed, 7.9 or 15.9 bandwidth accordingly.
- **EISA** - Extended Industry Standard Architecture:
32 bits, 8.3 MHz, 31.8 bandwidth, the attempt to compete with IBM's MCA
- **PCI** - Peripheral Component Interconnect:
32 bits, 33 MHz, 127.2 bandwidth
- **PCI-X** - Up to 133 MHz bus speed, 64 bits bandwidth, 1GB/sec throughput
- **AGP** - Accelerated Graphics Port:
32 bits, 66 MHz, 254,3 bandwidth

Major ports and connectors/interface

- **IDE** - Integrated Drive Electronics, also know as ATA, EIDE, Ultra ATA, Ultra DMA, most widely used interface for hard disks
- **PS/2 port** - mini Din plug with 6 pins for a mouse and keyboard
- **SCSI** - Small Computer System Interface, 80 - 640 Mbs, capable of handling internal/external peripherals
- **Serial Port** - adheres to RS-232c spec, uses DB9 or DB25 connector, capable of 115kb.sec speeds
- **Parallel port** - as know as printer port, enhanced types: ECP- extended capabilities port, EPP - enhanced parallel port
- **USB - universal serial bus**, two types: 1.0 and 2.0, hot plug-and-play, at 12MB/s, up to 127 devices chain. 2.0 data rate is at 480 bits/s.
- **Firewire** - high speed serial port, 400 MB/s, hot plug-and-play, 30 times faster than USB 1.0

Module 2 : Arithmetic and Logic Unit

In this Module, we have two lectures, viz.

1. [Arithmetic and logical operation and Hardware implementation](#)
2. [Implementation issues of some operation](#)

Click the proper link on the left side for the lectures

Arithmetic and logic Unit (ALU)

ALU is responsible to perform the operation in the computer.

The basic operations are implemented in hardware level. ALU is having collection of two types of operations:

- Arithmetic operations
- Logical operations

Consider an ALU having 4 arithmetic operations and 4 logical operation.

To identify any one of these four logical operations or four arithmetic operations, two control lines are needed. Also to identify the any one of these two groups- arithmetic or logical, another control line is needed. So, with the help of three control lines, any one of these eight operations can be identified.

Consider an ALU is having four arithmetic operations. Addition, subtraction, multiplication and division. Also consider that the ALU is having four logical operations: OR, AND, NOT & EX-OR.

We need three control lines to identify any one of these operations. The input combination of these control lines are shown below:

Control line C_2 is used to identify the group: logical or arithmetic, ie

$C_2 = 0$: arithmetic operation $C_2 = 1$: logical operation.

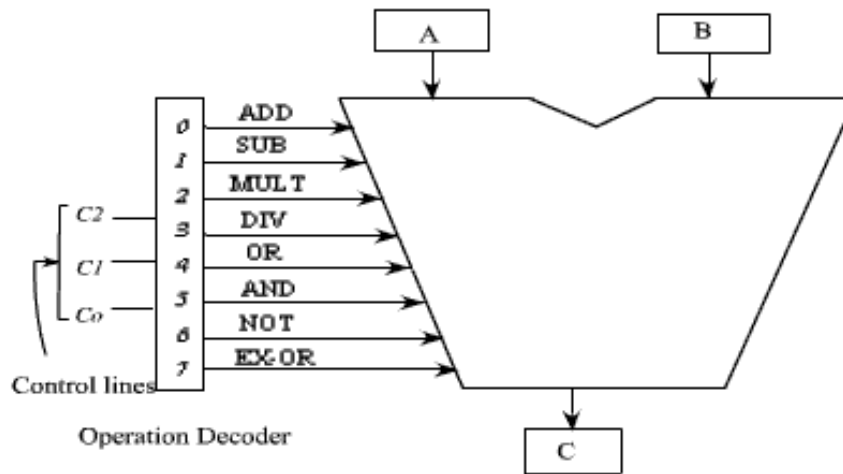
Control lines C_0 and C_1 are used to identify any one of the four operations in a group. One possible combination is given here.

C_1	C_0	Arithmetic $C_2 = 0$	Logical $C_2 = 1$
0	0	Addition	OR
0	1	Subtraction	AND
1	0	Multiplication	NOT
1	1	Division	EX-OR

Arithmetic and logic Unit (ALU)

[Print this page](#)[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

A 3×8 decode is used to decode the instruction. The block diagram of the ALU is shown in the figure.



Block Diagram of the ALU

[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

Arithmetic and logic Unit (ALU)[Print this page](#)[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

The ALU has got two input registers named as A and B and one output storage register, named as C. It performs the operation as:

$$C = A \text{ op } B$$

The input data are stored in A and B, and according to the operation specified in the control lines, the ALU perform the operation and put the result in register C.

As for example, if the contents of controls lines are, 000, then the decoder enables the addition operation and it activates the adder circuit and the addition operation is performed on the data that are available in storage register A and B . After the completion of the operation, the result is stored in register C.

We should have some hardware implementations for basic operations. These basic operations can be used to implement some complicated operations which are not feasible to implement directly in hardware.

[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

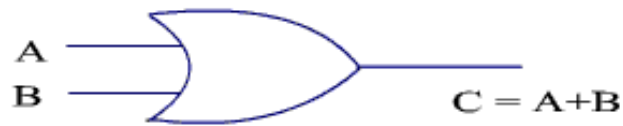
There are several logic gates exists in digital logic circuit. These logic gates can be used to implement the logical operation. Some of the common logic gates are mentioned here.

AND gate: The output is high if both the inputs are high.



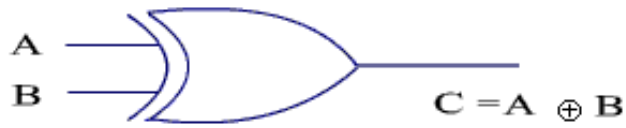
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR gate: The output is high if any one of the input is high.



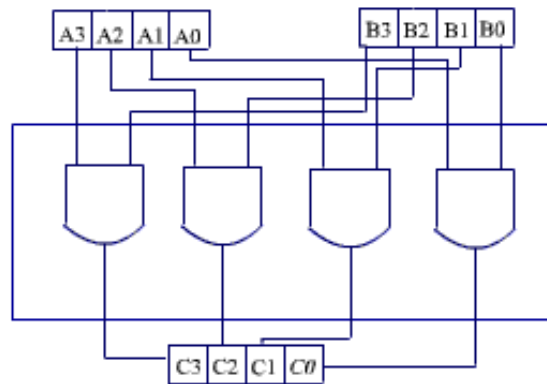
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

EX-OR gate: The output is high if either of the input is high.



A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

If we want to construct a circuit which will perform the AND operation on two 4-bit number, the implementation of the 4-bit AND operation is shown in the figure.



4-bit AND operator

Arithmetic Circuit

Binary Adder : Binary adder is used to add two binary numbers.

In general, the adder circuit needs two binary inputs and two binary outputs. The input variables designate the augends and addend bits; The output variables produce the sum and carry.

The binary addition operation of single bit is shown in the truth table

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

C: Carry Bit

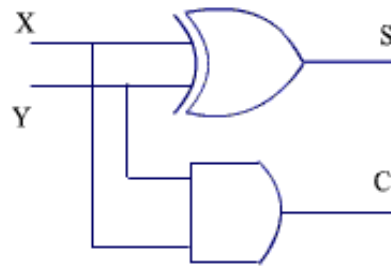
S: Sum Bit

The simplified sum of products expressions are

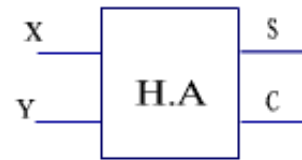
$$S = x'y + xy'$$

$$C = xy$$

The circuit implementation is



Circuit Diagram



Block Diagram

This circuit can not handle the carry input, so it is termed as **half adder**.

Full Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs.

Two of the input variables, denoted by x and y , represent the two bits to be added. The third input Z , represents the carry from the previous lower position.

The two outputs are designated by the symbols S for sum and C for carry.

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The simplified expression for S and C are

$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$C = xy + xz + yz$$

$$= xy + xy'z + x'yz$$

We may rearrange these two expressions as follows:

$$S = z \oplus (x \oplus y)$$

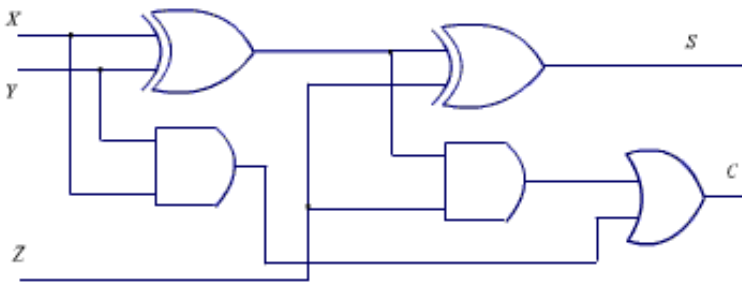
$$= z' (xy' + x'y) + z (xy' + x'y)'$$

$$= z' (xy' + x'y) + z (xy + x'y')$$

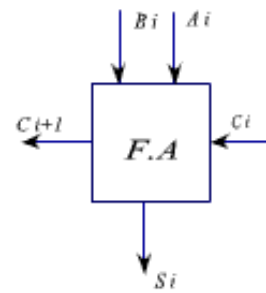
$$= xy'z' + x'yz' + xyz + x'y'z$$

$$C = z (xy' + x'y) + xy = xy'z + x'yz + xy$$

The circuit diagram full adder is shown in the figure.



Circuit Diagram



Block Diagram

n -such single bit full adder blocks are used to make n -bit full adder.

To demonstrate the binary addition of four bit numbers, let us consider a specific example.

Consider two binary numbers

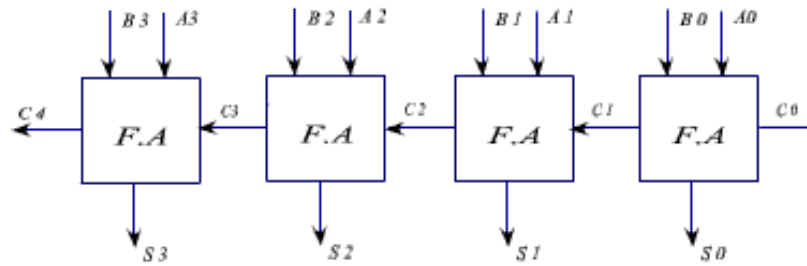
$$A = 1\ 0\ 0\ 1$$

$$B = 0\ 0\ 1\ 1$$

Subscript	i	3	2	1	0
Input carry	C_i	0	1	1	0
Augend	A_i	1	0	0	1
Addend	B_i	0	0	1	1
Sum	S_i	1	1	0	0
Output Carry	C_{i+1}	0	0	1	1

To get the four bit adder, we have to use 4 full adder block. The carry output the lower bit is used as a carry input to the next higher bit.

The circuit of 4-bit adder shown in the figure.



Binary subtractor : The subtraction operation can be implemented with the help of binary adder circuit, because

$$A - B = A + (-B)$$

We know that 2's complement representation of a number is treated as a negative number of the given number.

We can get the 2's complements of a given number by complementing each bit and adding 1 to it.

The circuit for subtracting $A-B$ consist of an added with inverter placed between each data input B and the corresponding input of the full adder. The input carry C_0 must be equal to 1 when performing subtraction.

The operation thus performed becomes A , plus the 1's complement of B , plus 1. This is equal to A plus 2's complement of B .

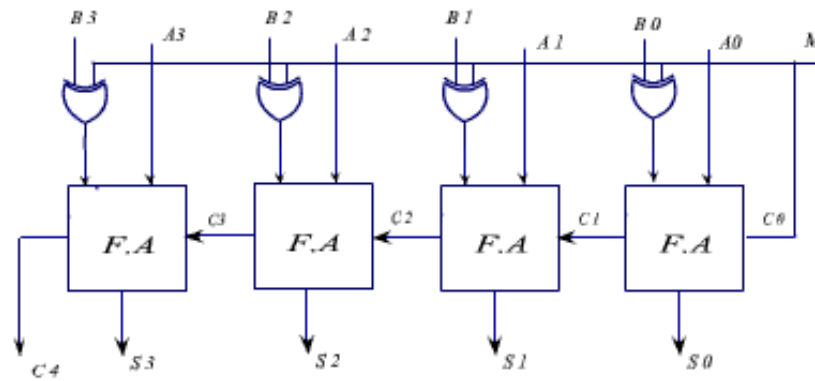
With this principle, a single circuit can be used for both addition and subtraction. The 4 bit adder subtractor circuit is shown in the figure. It has got one mode (M) selection input line, which will determine the operation,

Arithmetic and logic Unit (ALU)

[Print this page](#)[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

If $M = 0$, then $A + B$

If $M = 1$ then $A - B = A + (-B)$
 $= A + 1\text{'s complement of } B + 1$



4-bit adder subtractor

[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

The operation of OR gate:

$$x \oplus 0 = x$$

$$x \oplus 1 = x'$$

$$\text{if } M = 0, \quad B_i \oplus 0 = B_i$$

$$\text{if } M = 1, \quad B_i \oplus 1 = B_i'$$

Multiplication

Multiplication of two numbers in binary representation can be performed by a process of SHIFT and ADD operations. Since the binary number system allows only 0 and 1's, the digit multiplication can be replaced by SHIFT and ADD operation only, because multiplying by 1 gives the number itself and multiplying by 0 produces 0 only.

The multiplication process is illustrated with a numerical example.

$$\begin{array}{r}
 25 \\
 \times 19 \\
 \hline
 225 \\
 25 \\
 \hline
 475
 \end{array}$$

11001	Multiplicand
10011	Multiplier
<hr/>	
11001	
11001	
00000	
00000	
11001	
<hr/>	
111011011	Product

The process consists of looking at successive bits of the multiplier, least significant bit first. If the multiplier bit is a 1, the multiplicand is copied down, otherwise, zeros are copied down. The numbers copied down in successive lines are shifted one position to the left from the previous number. Finally, the numbers are added and their sum forms the product.

When multiplication is implemented in a digital computer, the process is changed slightly.

Instead of providing registers to store and add simultaneously as many binary numbers as there are bits in the multiplier, it is convenient to provide an adder for the summation of only two binary numbers and successively accumulate the partial products in a register. It will reduce the requirements of registers.

Instead of shifting the multiplicand to the left, the partial product is shifted to right.

When the corresponding bit of the multiplier is 0, there is no need to add all zeros to the partial product.

An algorithm to multiply two binary numbers. Consider that the ALU does not provide the multiplication operation, but it is having the addition operation and shifting operation. Then we can write a micro program for multiplication operation and provide the micro program code in memory. When a multiplication operation is encountered, it will execute this micro code to perform the multiplication.

Implemental issue of some operations[Print this page](#)[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

The micro code is nothing but the collection of some instructions. ALU must have those operation; otherwise we must have micro code again for those operations which are not supported in ALU.

Consider a situation such that we do not have the multiplication operation in a primitive computer. Is it possible to perform the multiplication. Of course, yes, provided the addition operation is available.

We can perform the multiplication with the help of repeated addition method; for example, if we want to multiply 4 by 5 (4×5), then simply add 4 five times to get the result.

If it is possible by addition operation, then why we need a multiplication operation.

Consider a machine, which can handle 8 bit numbers, then we can represent number from 0 to 255. If we want to multiply 175×225 , then there will be at least 175 addition operation.

But if we use the multiplication algorithm that involves shifting and addition, it can be done in 8 steps, because we are using an 8-bit machine.

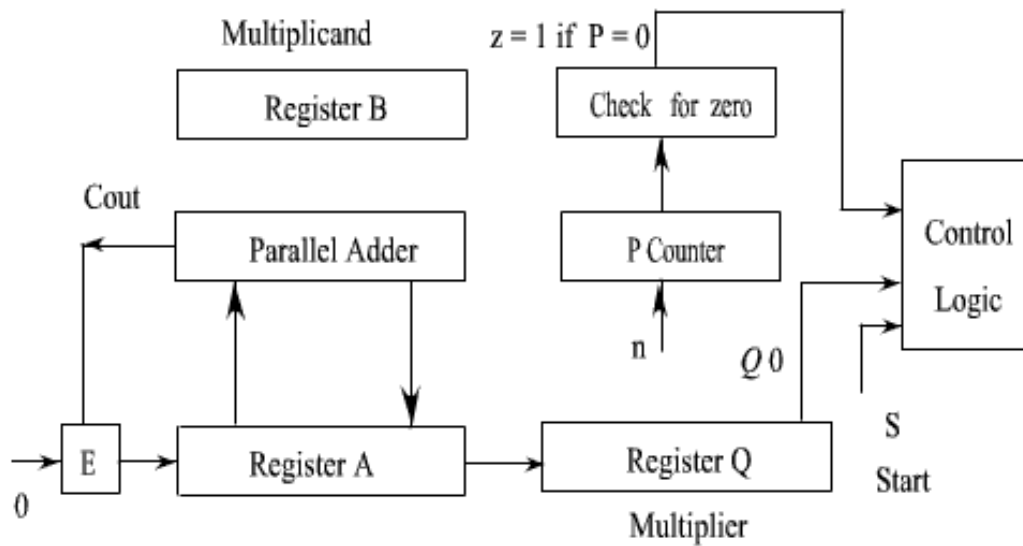
Again, the micro program execution is slightly slower, because we have to access the code from micro controller memory, and memory is a slower device than CPU.

It is possible to implement the multiplication algorithm in hardware.

[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

Binary Multiplier, Hardware Implementation

The block diagram of binary multiplier is shown in the figure.



Block diagram of binary multiplier

Implemental issue of some operations[Print this page](#)[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

The multiplicand is stored in register B and the multiplier is stored in register Q.

The partial product is formed in register A and stored in A and Q

The counter P is initially set to a number equal to the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product. When the content of the counter reaches zero, the product is formed and the process stops.

Initially, the multiplicand is in register B and the multiplier in Q. The register A is reset to 0.

The sum of A and B forms a partial product- which is transferred to the EA register.

Both partial product and multiplier are shifted to the right. The least significant bit of A is shifted into the most significant position of Q; and 0 is shifted into E.

After the shift, one bit of the partial product is shifted into Q, pushing the multiplier bits one position to the right.

The right most flip flop in register Q, designated by Q_0 will hold the bit of the multiplier which must be inspected next. If the content of this bit is 0, then it is not required to add the multiplicand, only shifting is needed. If the content of this bit is 1, then both addition and shifting are needed.

[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

Implemental issue of some operations[Print this page](#)[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

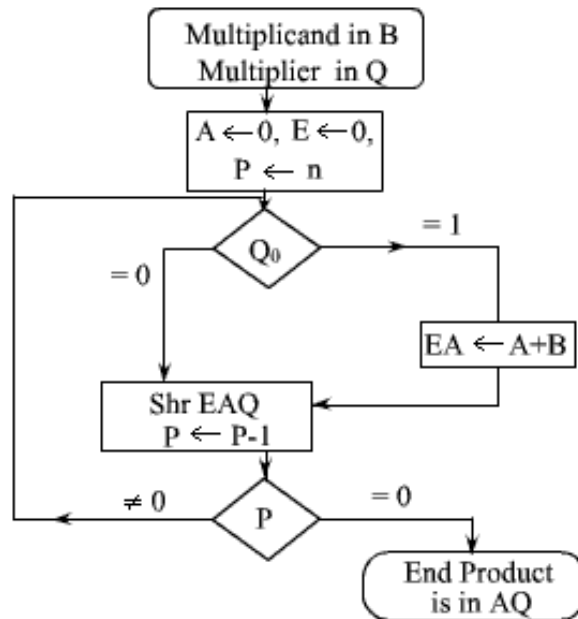
After each shifter, value of counter P is decremented and the process continues till the counter value becomes 0.

The final result is available in (EAQ) registers combination.

To control the operation, it is required to design the appropriate control logic that is shown in the block diagram.

[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

The flow chart of the multiplication operation is given in the figure.



Implemental issue of some operations

[Print this page](#)
[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

The working of multiplication algorithm is shown here with the help of an example.

Multiplicand $B = 11001$

	E	A	Q	P
Multiplier in Q	0	00000	10011	5
$Q_0 = 1$, Add B		11001		

first partial product	0	11001		
Shr EAQ	0	01100	11001	4
$Q_0 = 1$, Add B		11001		

Second partial product	1	00101		
Shr EAQ	0	10010	11100	3
$Q_0 = 0$, Shr EAQ	0	01001	01110	2

[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

Implemental issue of some operations

[Print this page](#)
[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)

$Q_0 = 0$, Shr EAQ 0 0 1 0 0 1 0 1 1 1 0 1

$Q_0 = 1$, Add B 1 1 0 0 1

fifth partial product 0 1 1 1 0 1

Shr EAQ 0 0 1 1 1 0 1 1 0 1 1 0

Stop

Final products: 0 1 1 1 0 1 1 0 1 1

[<< Previous](#) | [First](#) | [Last](#) | [Next >>](#)