17CSU102 **COMPUTER SYSTEM ARCHITECTURE** **Semester – I**
4H – 4C

Instruction Hours / week: L: 4 T: 0 P: 0 **Marks:** Int : **40** Ext : **60** Total: **100**

SCOPE

Computer System Architecture deals with the architecture of computer systems with its various processing units and also the performance measurement of the computer system. This course is designed to provide a comprehensive introduction to digital logic design leading to the ability to understand number system representations, binary codes, binary arithmetic and Boolean algebra, and its relevance to digital logic design.

OBJECTIVE

- To enable the students to gain knowledge on the architecture of modern computer.
- To understand how computer stores positive and negative numbers and to perform arithmetic operation of positive and negative numbers.
- To learn Cache memory and its importance

UNIT -I Introduction

Logic gates, Boolean algebra, circuit simplification, combinational circuits: Adders and Subtractors –Multiplexers and De multiplexers – Encoders and Decoders-sequential circuits: Flip Flop's, registers, counters and memory units.

UNIT -II Data Representation and Basic Computer Arithmetic

Number systems, complements, fixed and floating point representation, character representation, addition, subtraction, magnitude comparison, multiplication and division algorithms for integers

UNIT –III Basic Computer Organization and Design

Computer registers, bus system, instruction set, timing and control, instruction cycle, memory reference, input-output and interrupt, Interconnection Structures, Bus interconnection design of basic computer.

UNIT-IV Central Processing Unit

Register organization, arithmetic and logical micro-operations, stack organization, micro programmed control, Instruction formats, addressing modes, instruction codes, machine language, assembly language, input output programming, RISC, CISC architectures, pipelining and parallel architecture.

UNIT –V Memory and Input-Output Organization

Cache memory, Associative memory, mapping Input / Output: External Devices, I/O Modules , Programmed I/O, Interrupt-Driven I/O, Direct Memory Access, I/O Channels.

Suggested Readings:

1. Dos'Reis, A. J. (2009). Assembly Language and Computer Architecture using C++ and JAVA. Course Technology
2. Stallings, W. (2010). Computer Organization and Architecture Designing for Performance (8th ed.) New Delhi: Prentice Hall of India,
3. Mano, M.M. (2013). Digital Design, New Delhi: Pearson Education Asia.
4. Carl Hamacher. (2012). Computer Organization (5th ed.). New Delhi: McGrawHill.

Class : I B.Sc. CS Semester-I

Computer Systems Architecture

Sno	Lecture Duration (Hours)	Topics to be Covered	Text Book/Ref. Book/WebSite
UNIT I			
Logic Gates			
1	1	Introduction: Logic Gates	T3 Pg No:1-4,58-62
2	1	Boolean Algebra-	T3 Pg No: 36-48
3	1	Circuit Simplification	T3 Pg No: 114-116
4	1	Combinational Circuits Adders and Subtractor	T3 Pg No: 116-123
5	1	Adders	
6	1	Subtractor	
7	1	Multiplexers and De-multiplexers	T3 Pg No: 48-50
8	1	Demultiplexers	
9	1	Encoders and Decoders	T3 Pg No: 43-48
10	1	Decoders	
11	1	Sequential Circuits	T3 Pg No: 28-29
12	1	Flip-Flops, registers	T3 Pg No:22-27,50-56
13	1	Register	
14	1	Counters and memory units	T3 Pg No:56-58,58-62
15	1	Recapitalization and Discussion of important questions	
Total no.of hours planned for unit-I			**15**
UNIT II			
Dta Representation and Basic Computer Architecture			
1	1	**Numbers Systems**	T3 Pg No: 67-74
2	1	Complements	T3 Pg No: 74-77
3	1	Fixed and Floating point Representation	T3 Pg No: 77-84
4	1	Character representation	T3 Pg No: 204-205
5	1	Addition, Subtraction	T3 Pg No: 334-336
6	1	Magnitude Comparison	T3 Pg No: 336-338
7	1	Multiplication algorithms for integers	T3 Pg No: 340-348
8	1	Division algorithms for integers	T3 Pg No: 348-353
9	1	Recapitalization and Discussion of important questions	
Total no. of periods planned for unit-II			**9**
UNIT III			
Basic Computer Organization and Design			
1	1	Computer registers, bus systems	T1 Pg No:127-129
2	1	Instruction set	T1 Pg No:129-131
3	1	timing and control	R1 Pg No:39
4	1	Instruction cycle	T1 Pg No:135-139

Computer Systems Architecture

5	1	memory reference	T1 Pg No:139-145
6	1	input-output and interrupt	T1 Pg No:145-149
7	1	interconnection structures	T1 Pg No:150-156
8	1	bus interconnection design of computers	T1 Pg No:130-135
9	1	Recapitalization and Discussion of important questions	
Total no.of hours planned for unit-IV			**9**
UNIT IV			
Central Processing Unit			
1	1	Register Organization	T1 Pg No: 241-245
		arithmetic and logical micro-operations	T1 Pg No:246-247
2	1	stack organization	T1 Pg No:247-257
		micro programmed control	T1 Pg No:255-259
3	1	instructions format, addressing modes	T1 Pg No:260-263
4	1	instruction codes, machine language, assembly language	T1 Pg No:179-190
5	1	input, output programming	R1 Pg No:16-18
6	1	RISC,CISC Architectures	T1 Pg No: 282-285
7	1	Pipelining and architecture	T1 Pg No: 302-318
8	1	Recapitalization and Discussion of important questions	
Total no. of periods planned for unit-IV			**7**
UNIT V			
Memory and Input-output Organization			
1	1	Cache memory	T1 Pg No:445-448
		Associative memory	T1 Pg No:456-459
2	1	mapping -input/output: external devices	T1 Pg No:464-465
3	1	I/O Modules	W1
4	1	Programmed I/O	W1
5	1	Interrupt Driven I/O	W1
6	1	Direct Memory Access	T1 Pg No:408-412
7	1	I/O Channels	T1 Pg No:415 420
8	1	Recapitalization and Discussion of important questions	
9	1	Discussion of Previous year ESE question paper1	
10	1	Discussion of Previous year ESE question paper2	
11	1	Discussion of Previous year ESE question paper3	
Total no.of hours planned for unit-V and Discussion of previous year ESE question =11			
TOTAL NO. OF HOURS ALLOTTED 60			

TEXT BOOKS:

1. Carl Hamacher ,(2012). Computer Organization,(5th ed.). McGraw Hill

Computer Systems Architecture

2. Dos Reis,A.J.,(2009), Assembly Language and Computer Architecture using C++ and JAVA Technology.
3. M.Mories Mano(2013) Computer Systems Architecture,(3rd ed.), Prentice Hall of India
4. Stalings, W., (2010). Computer Organization and Architecture Designing for Performance,(8th ed.), Prentice Hall of India

WEBSITE

1. https://www.falstad.com

UNIT I

Basic Gates

Boolean functions may be practically implemented by using electronic gates. The following points are important to understand.

- Electronic gates require a power supply.
- Gate **INPUTS** are driven by voltages having two nominal values, e.g. 0V and 5V representing logic 0 and logic 1 respectively.
- The **OUTPUT** of a gate provides two nominal values of voltage only, e.g. 0V and 5V representing logic 0 and logic 1 respectively. In general, there is only one output to a logic gate except in some special cases.
- There is always a time delay between an input being applied and the output responding.

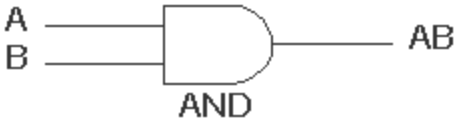
Truth Tables

Truth tables are used to help show the function of a logic gate. If you are unsure about truth tables and need guidance on how go about drawing them for individual gates or logic circuits then use the truth table section link.

Logic gates

Digital systems are said to be constructed by using logic gates. These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates. The basic operations are described below with the aid of truth tables.

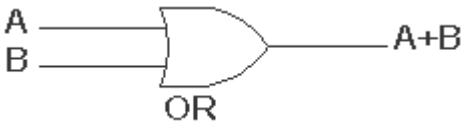
AND gate



2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is sometimes omitted i.e. AB

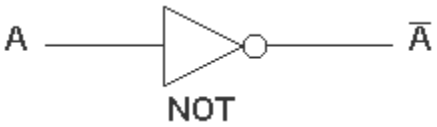
OR gate



2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

The OR gate is an electronic circuit that gives a high output (1) if **one or more** of its inputs are high. A plus (+) is used to show the OR operation.

NOT gate



NOT gate	
A	$\overline{A}$
0	1
1	0

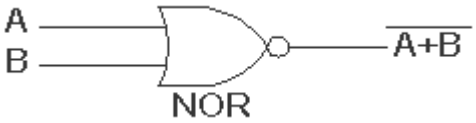
The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an *inverter*. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs

NAND gate

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if **any** of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

NOR gate



2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

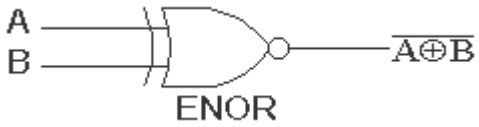
This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if **any** of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

EXOR gate

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

The '**Exclusive-OR**' gate is a circuit which will give a high output if **either, but not both**, of its two inputs are high. An encircled plus sign (⊕) is used to show the EOR operation.

EXNOR gate



2 Input EXNOR gate		
A	B	$\overline{A⊕B}$
0	0	1
0	1	0
1	0	0
1	1	1

The '**Exclusive-NOR**' gate circuit does the opposite to the EOR gate. It will give a low output if **either, but not both**, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.

The NAND and NOR gates are called *universal functions* since with either one the AND and OR functions and NOT can be generated.

Note:

A function in *sum of products* form can be implemented using NAND gates by replacing all AND and OR gates by NAND gates. A function in *product of sums* form can be implemented using NOR gates by replacing all AND and OR gates by NOR gates.

Table 1: Logic gate symbols

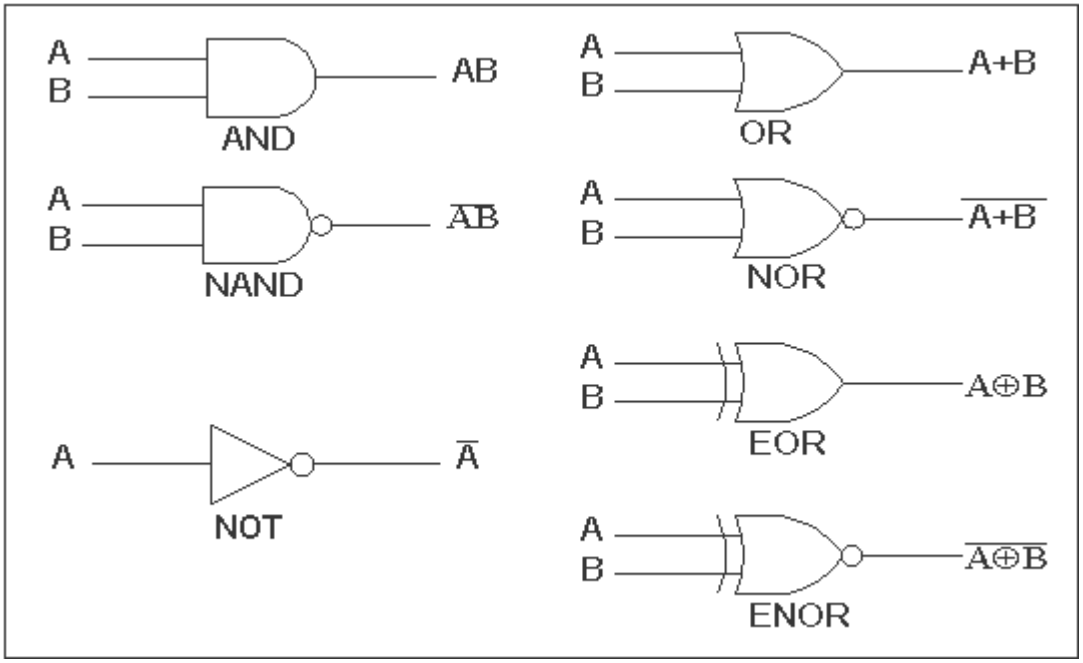


Table 2 is a summary truth table of the input/output combinations for the NOT gate together with all possible input/output combinations for the other gate functions. Also note that a truth table with 'n' inputs has 2^n rows. You can compare the outputs of different gates.

Table 2: Logic gates representation using the Truth table

NOT gate	
A	$\overline{A}$
0	1
1	0

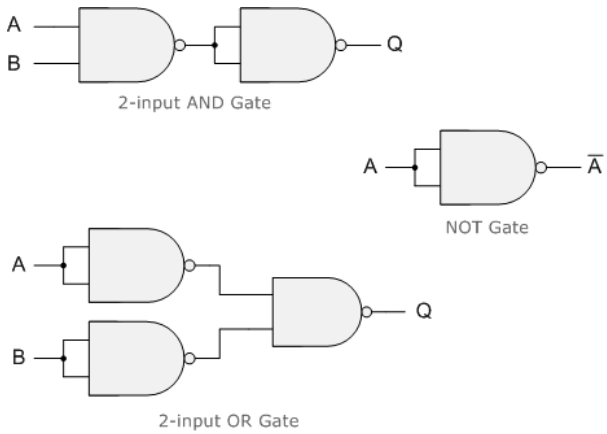
INPUTS		OUTPUTS					
A	B	AND	NAND	OR	NOR	EXOR	EXNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

The "Universal" NAND Gate

The **Logic NAND Gate** is generally classed as a "Universal" gate because it is one of the most commonly used logic gate types. NAND gates can also be used to produce any other type of logic gate function, and in practice the NAND gate forms the basis of most practical logic circuits. By connecting them together in various combinations the three basic gate types of AND, OR and NOT function can be formed using only NAND's, for example.

Various Logic Gates using only NAND Gates

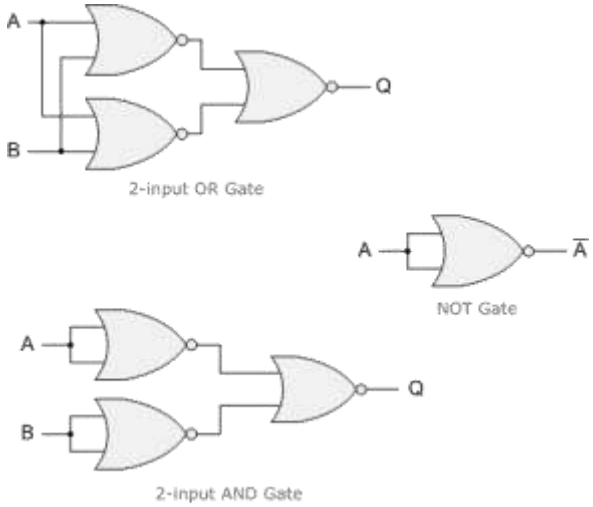
As well as the three common types above, Ex-Or, Ex-Nor and standard NOR gates can be formed using just individual NAND gates.



The "Universal" NOR Gate

Like the NAND gate seen in the last section, the NOR gate can also be classed as a "Universal" type gate. NOR gates can be used to produce any other type of logic gate function just like the NAND gate and by connecting them together in various combinations the three basic gate types of AND, OR and NOT function can be formed using only NOR's, for example.

Various Logic Gates using only NOR Gates



As well as the three common types above, Ex-Or, Ex-Nor and standard NOR gates can also be formed using just individual NOR gates.

Boolean arithmetic

Let us begin our exploration of Boolean algebra by adding numbers together:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 1

The first three sums make perfect sense to anyone familiar with elementary addition. The last sum, though, is quite possibly responsible for more confusion than any other single statement in digital electronics, because it seems to run contrary to the basic principles of mathematics. Well, it *does* contradict principles of addition for real numbers, but not for Boolean numbers. Remember that in the world of Boolean algebra, there are only two possible values for any quantity and for any arithmetic operation: 1 or 0. There is no such thing as"2" within the scope of Boolean values. Since the sum"1 + 1" certainly isn't 0, it must be 1 by process of elimination.

It does not matter how many or few terms we add together, either. Consider the following sums:

0 + 1 + 1 = 1

0 + 1 + 1 + 1 = 1

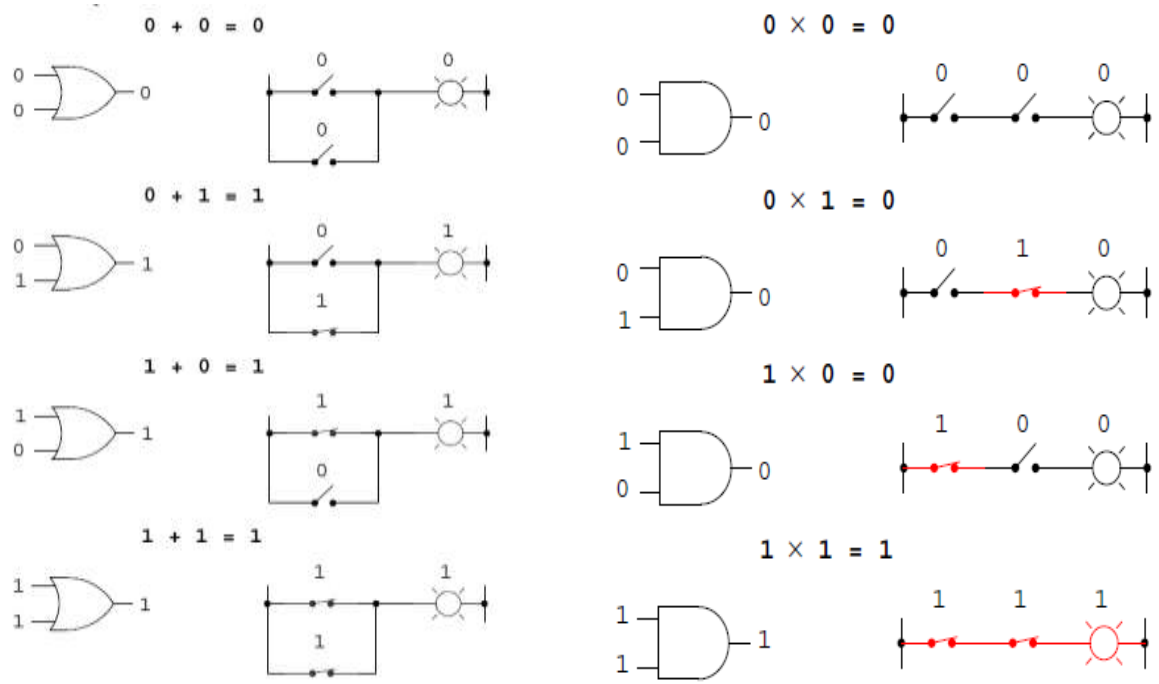
1 + 0 + 1 + 1 + 1 = 1

1 + 1 + 1 = 1

Take a close look at the two-term sums in the first set of equations. Does that pattern look familiar to you? It should! It is the same pattern of 1's and 0's as seen in the truth table for an

OR gate. In other words, Boolean addition corresponds to the logical function of an"OR" gate, as well as to parallel switch contacts:

There is no such thing as subtraction in the realm of Boolean mathematics. Subtraction implies the existence of negative numbers: 5 - 3 is the same thing as 5 + (-3), and in Boolean algebra negative quantities are forbidden. There is no such thing as division in Boolean mathematics, either, since division is really nothing more than compounded subtraction, in the same way that multiplication is compounded addition.



Multiplication is valid in Boolean algebra, and thankfully it is the same as in real-number algebra: anything multiplied by 0 is 0, and anything multiplied by 1 remains unchanged:

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

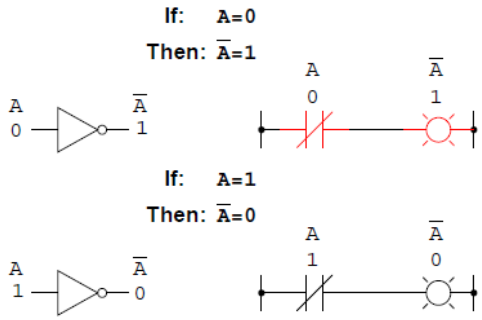
This set of equations should also look familiar to you: t is the same pattern found in the truth table for an AND gate. In other words, Boolean multiplication corresponds to the logical function of an"AND" gate, as well as to series switch contacts:

Like"normal" algebra, Boolean algebra uses alphabetical letters to denote variables. Unlike "normal" algebra, though, Boolean variables are always CAPITAL letters, never lowercase. Because they are allowed to possess only one of two possible values, either 1 or 0, each and every variable has a *complement*: the opposite of its value. For example, if variable"A" has a value of 0, and then the complement of A has a value of 1. Boolean notation uses a bar above the variable character to denote complementation, like this:

If: A=0, Then: A=1

If: A=1 Then: A=0

In written form, the complement of"A" denoted as"A-not" or"A-bar". Sometimes a" prime" symbol is used to represent complementation. For example, A' would be the complement of A, much the same as using a prime symbol to denote differentiation in calculus rather than the fractional notation d/dt. Usually, though, the"bar" symbol finds more widespread use than the "prime" symbol, for reasons that will become more apparent later in this chapter.
Boolean complementation finds equivalency in the form of the NOT gate, or a normally closed switch or relay contact:



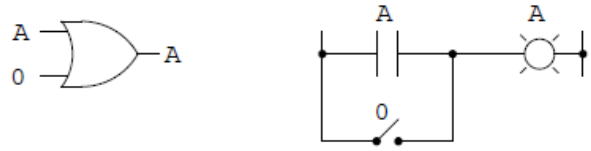
The basic definition of Boolean quantities has led to the simple rules of addition and multiplication, and has excluded both subtraction and division as valid arithmetic operations. We have a symbology for denoting Boolean variables, and their complements.

• **REVIEW:**
• Boolean addition is equivalent to the *OR* logic function, as well as *parallel* switch contacts.
• Boolean multiplication is equivalent to the *AND* logic function, as well as *series* switch contacts.
• Boolean complementation is equivalent to the *NOT* logic function, as well as *normally closed* relay contacts.

Basic Laws

In mathematics, an *identity* is a statement true for all possible values of its variable or variables. The algebraic identity of x + 0 = x tells us that anything (x) added to zero equals the original "anything," no matter what value that "anything" (x) may be. Like ordinary algebra, Boolean algebra has its own unique identities based on the bivalent states of Boolean variables. The first Boolean identity is that the sum of anything and zero is the same as the original"anything." This identity is no different from its real-number algebraic equivalent:

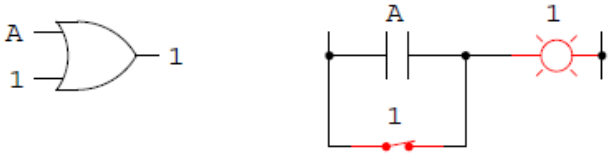
A + 0 = A



No matter what the value of A, the output will always be the same: when A=1, the output will also be 1; when A=0, the output will also be 0.

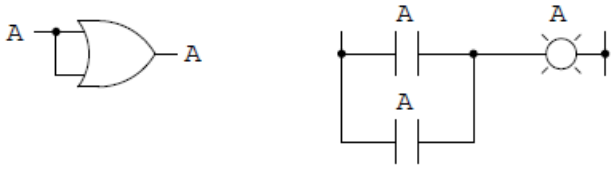
The next identity is most definitely *different* from any seen in normal algebra. Here we discover that the sum of anything and one is one:

A + 1 = 1



No matter what the value of A, the sum of A and 1 will always be 1. In a sense, the"1" signal *overrides* the effect of A on the logic circuit, leaving the output fixed at a logic level of 1. Next, we examine the effect of adding A and A together, which is the same as connecting both inputs of an OR gate to each other and activating them with the same signal:

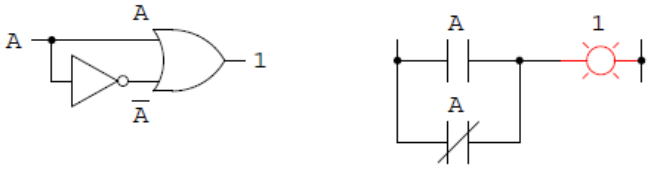
A + A = A



In real-number algebra, the sum of two identical variables is twice the original variable's value (x + x = 2x), but remember that there is no concept of"2" in the world of Boolean math, only 1 and 0, so we cannot say that A + A = 2A. Thus, when we add a Boolean quantity to itself, the sum is equal to the original quantity: 0 + 0 = 0, and 1 + 1 = 1.

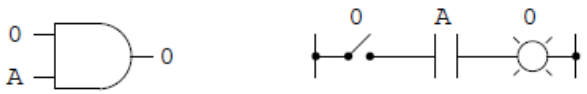
Introducing the uniquely Boolean concept of complementation into an additive identity, we find an interesting effect. Since there must be one"1" value between any variable and its complement, and since the sum of any Boolean quantity and 1 is 1, the sum of a variable and Its complement must be 1:

A + $\overline{A}$ = 1

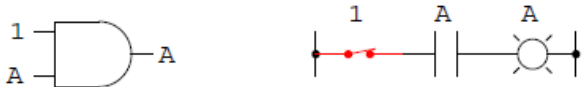


Just as there are four Boolean additive identities (A+0, A+1, A+A, and A+A'), so there are also four multiplicative identities: Ax0, Ax1, AxA, and AxA'. Of these, the first two are no different from their equivalent expressions in regular algebra:

0A = 0

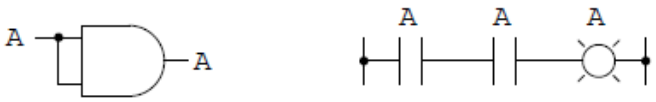


1A = A



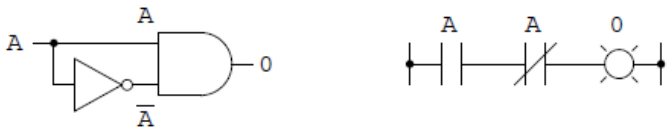
The third multiplicative identity expresses the result of a Boolean quantity multiplied by itself. In normal algebra, the product of a variable and itself is the *square* of that variable (3 x 3 = 32 = 9). However, the concept of"square" implies a quantity of 2, which has no meaning in Boolean algebra, so we cannot say that A x A = A2. Instead, we find that the product of a Boolean quantity and itself is the original quantity, since 0 x 0 = 0 and 1 x 1 = 1:

AA = A



The fourth multiplicative identity has no equivalent in regular algebra because it uses the complement of a variable, a concept unique to Boolean mathematics. Since there must be one "0" value between any variable and its complement, and since the product of any Boolean quantity and 0 is 0, the product of a variable and its complement must be 0:

A$\overline{A}$ = 0



To summarize, then, we have four basic Boolean identities for addition and four for multiplication:

Additive

A + 0 = A

$$A + 1 = 1$$

$$A + A = A$$

$$A + A = 1$$

Multiplicative

$$0A = 0$$

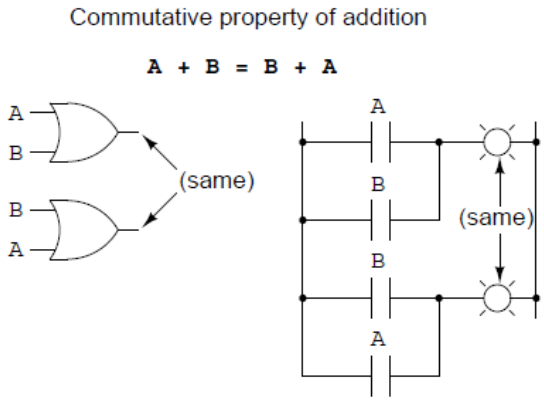
$$1A = A$$

$$AA = A$$

$$AA = 0$$

Basic Boolean algebraic identities

Another identity having to do with complementation is that of the *double complement*: a variable inverted twice. Complementing a variable twice (or any even number of times) results in the original Boolean value. This is analogous to negating (multiplying by -1) in real-number algebra: an even number of negations cancel to leave the original value:

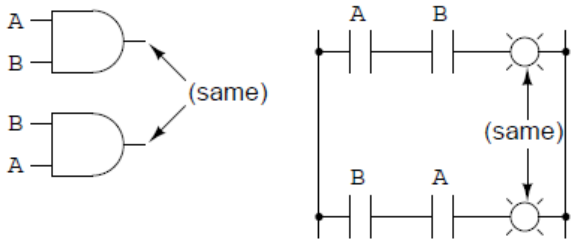


Boolean algebraic properties

Another type of mathematical identity, called a ”property” or a ”law,” describes how differing variables relate to each other in a system of numbers. One of these properties is known as the *commutative property*, and it applies equally to addition and multiplication. In essence, the commutative property tells us we can reverse the order of variables that are either added together or multiplied together without changing the truth of the expression:

Commutative property of multiplication

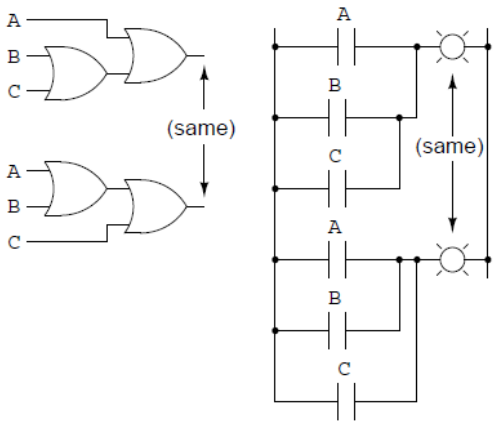
AB = BA



Along with the commutative properties of addition and multiplication, we have the *associative property*, again applying equally well to addition and multiplication. This property tells us we can associate groups of added or multiplied variables together with parentheses without altering the truth of the equations.

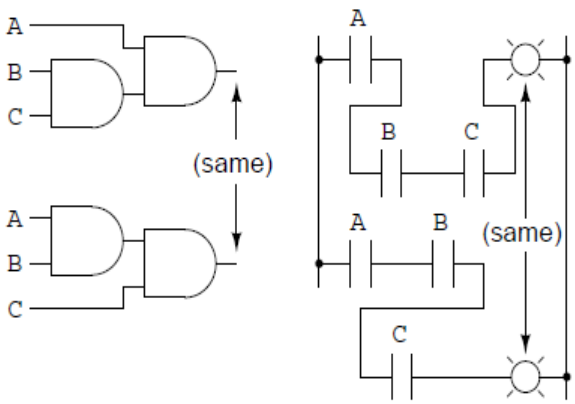
Associative property of addition

A + (B + C) = (A + B) + C

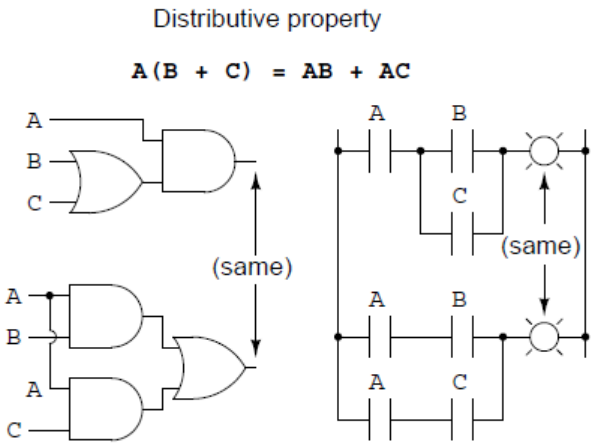


Associative property of multiplication

A(BC) = (AB)C



Lastly, we have the *distributive property*, illustrating how to expand a Boolean expression formed by the product of a sum, and in reverse shows us how terms may be factored out of Boolean sums-of-products:



To summarize, here are the three basic properties: commutative, associative, and distributive.

Basic Boolean algebraic properties

$$A(B + C) = AB + AC$$

Additive

$$A + (B + C) = (A + B) + C$$

$$A + B = B + A$$

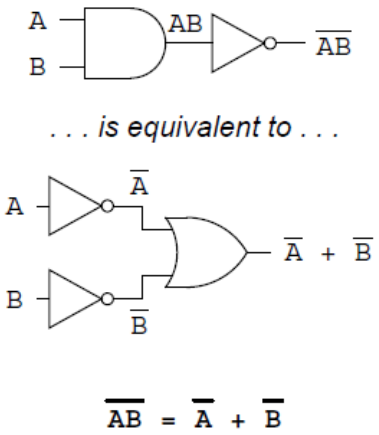
Multiplicative

$$A(BC) = (AB)C$$

$$AB = BA$$

DeMorgan's Theorems

A mathematician named DeMorgan developed a pair of important rules regarding group complementation in Boolean algebra. By *group* complementation, I'm referring to the complement of a group of terms, represented by a long bar over more than one variable. You should recall from the chapter on logic gates that inverting all inputs to a gate reverses that gate's essential function from AND to OR, or vice versa, and also inverts the output. So, an OR gate with all inputs inverted (a Negative-OR gate) behaves the same as a NAND gate, and an AND gate with all inputs inverted (a Negative-AND gate) behaves the same as a NOR gate. DeMorgan's theorems state the same equivalence in "backward" form: that inverting the output of any gate results in the same function as the opposite type of gate (AND vs. OR) with inverted inputs:

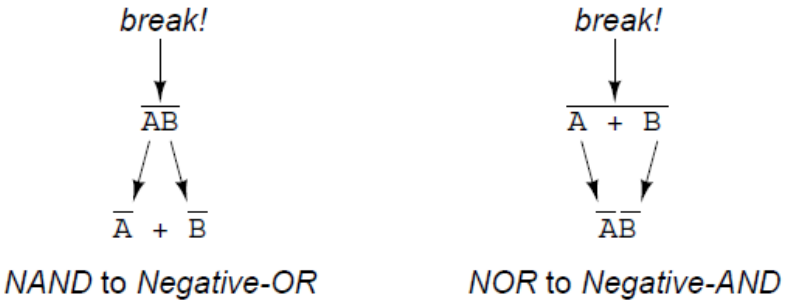


$$\overline{AB} = \overline{A} + \overline{B}$$

A long bar extending over the term AB acts as a grouping symbol, and as such is entirely different from the product of A and B independently inverted. In other words, (AB)' is not equal to A'B'. Because the "prime" symbol (') cannot be stretched over two variables like a bar can, we are forced to use parentheses to make it apply to the whole term AB in the previous sentence. A bar, however, acts as its own grouping symbol when stretched over more than one variable. This has profound impact on how Boolean expressions are evaluated and reduced, as we shall see.

DeMorgan's theorem may be thought of in terms of *breaking* a long bar symbol. When a long bar is broken, the operation directly underneath the break changes from addition to multiplication, or vice versa, and the broken bar pieces remain over the individual variables. To illustrate:

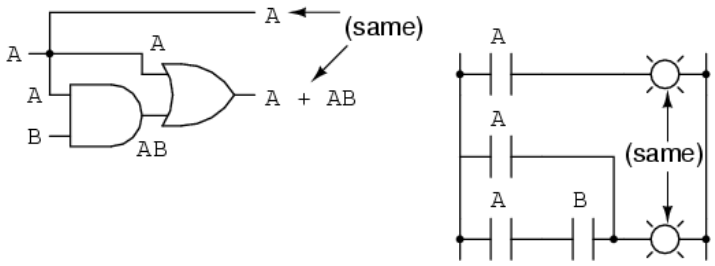
DeMorgan's Theorems



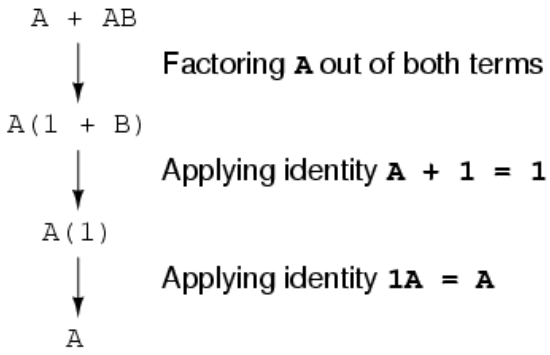
Boolean algebra finds its most practical use in the simplification of logic circuits. If we translate a logic circuit's function into symbolic (Boolean) form, and apply certain algebraic rules

to the resulting equation to reduce the number of terms and/or arithmetic operations, the simplified equation may be translated back into circuit form for a logic circuit performing the same function with fewer components. If equivalent function may be achieved with fewer components, the result will be increased reliability and decreased cost of manufacture.
To this end, there are several rules of Boolean algebra presented in this section for use in reducing expressions to their simplest forms. The identities and properties already reviewed in this chapter are very useful in Boolean simplification, and for the most part bear similarity to many identities and properties of "normal" algebra. However, the rules shown in this section are all unique to Boolean mathematics.

A + AB = A



This rule may be proven symbolically by factoring an "A" out of the two terms, then applying the rules of A + 1 = 1 and 1A = A to achieve the final result:

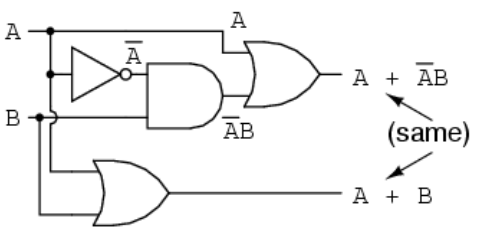


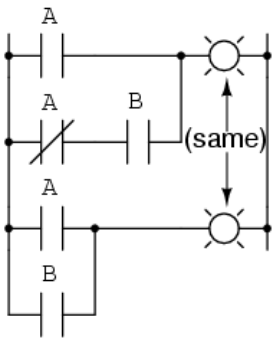
Please note how the rule A + 1 = 1 was used to reduce the (B + 1) term to 1. When a rule like "A + 1 = 1" is expressed using the letter "A", it doesn't mean it only applies to expressions containing "A". What the "A" stands for in a rule like A + 1 = 1 is *any* Boolean variable or collection of variables. This is perhaps the most difficult concept for new students to master in Boolean simplification: applying standardized identities, properties, and rules to expressions not in standard form.

For instance, the Boolean expression ABC + 1 also reduces to 1 by means of the "A + 1 = 1" identity. In this case, we recognize that the "A" term in the identity's standard form can represent the entire "ABC" term in the original expression.

The next rule looks similar to the first one shown in this section, but is actually quite different and requires a more clever proof:

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





$A + \overline{A}B$

↓ Applying the previous rule to expand **A** term

A + AB = A

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

↓ Factoring **B** out of 2nd and 3rd terms

$A + B(A + \overline{A})$

↓ Applying identity $\mathbf{A + \overline{A} = 1}$

$A + B(1)$

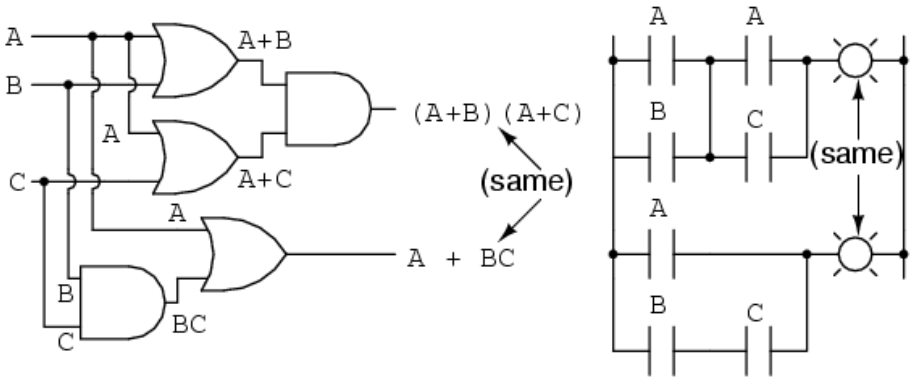
↓ Applying identity **1A = A**

$A + B$

Note how the last rule (A + AB = A) is used to "un-simplify" the first "A" term in the expression, changing the "A" into an "A + AB". While this may seem like a backward step, it certainly helped to reduce the expression to something simpler! Sometimes in mathematics we must take "backward" steps to achieve the most elegant solution. Knowing when to take such a step and when not to is part of the art-form of algebra, just as a victory in a game of chess almost always requires calculated sacrifices.

Another rule involves the simplification of a product-of-sums expression:

(A + B)(A + C) = A + BC



And, the corresponding proof:

(A + B)(A + C)

↓ Distributing terms

AA + AC + AB + BC

↓ Applying identity **AA = A**

A + AC + AB + BC

↓ Applying rule **A + AB = A** to the A + AC term

A + AB + BC

↓ Applying rule **A + AB = A** to the A + AB term

A + BC

To summarize, here are the three new rules of Boolean simplification expounded in this section:

Useful Boolean rules for simplification

$$A + AB = A$$

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

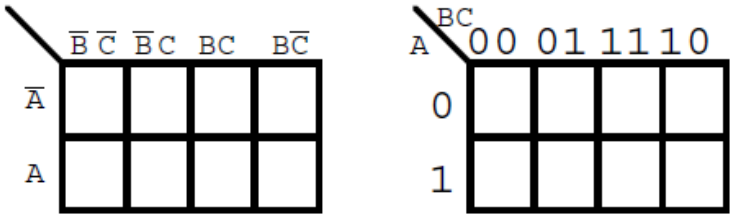
$$(A + B)(A + C) = A + BC$$

Karnaugh map

The Karnaugh map, like Boolean algebra, is a simplification tool applicable to digital logic. The Karnaugh Map will simplify logic faster and more easily in most cases.

Boolean simplification is actually faster than the Karnaugh map for a task involving two or fewer Boolean variables. It is still quite usable at three variables, but a bit slower. At four

input variables, Boolean algebra becomes tedious. Karnaugh maps are both faster and easier. Karnaugh maps work well for up to six input variables, are usable for up to eight variables. For more than six to eight variables, simplification should be by *CAD* (computer automated design).



Relationship between a Karnaugh Map and a Truth Table

Each row in the table (or minterm) is equivalent to a a cell on the Karnaugh Map.

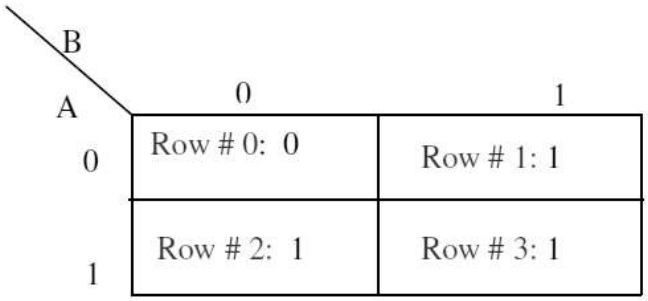
Example #1:

Here is a two-input truth table for a digital circuit:

Row Inputs Output

Row	**Inputs**		**Output**
	A	**B**	**F**
Row # 0	0	0	0
Row # 1	0	1	1
Row # 2	1	0	1
Row # 3	1	1	1

The corresponding K-map is:



Example #2:

Here is a three-input truth table for a digital circuit:

Row	Inputs			Output
	A	B	C	F
Row # 0	0	0	0	0
Row # 1	0	0	1	1
Row # 2	0	1	0	1
Row # 3	0	1	1	1
Row # 4	1	0	0	1
Row # 5	1	0	1	1
Row # 6	1	1	0	0
Row # 7	1	1	1	1

The corresponding K-map is:

C \ AB	00	01	11	10
0	Row # 0 0	Row # 2 1	Row # 6 0	Row # 4 1
1	Row # 1 1	Row # 3 1	Row # 7 1	Row # 5 1

Example #3:

Here is a four-input truth table for a digital circuit:

Row	Inputs				Output
	A	B	C	D	F
Row # 0	0	0	0	0	0
Row # 1	0	0	0	1	1
Row # 2	0	0	1	0	1
Row # 3	0	0	1	1	1
Row # 4	0	1	0	0	1
Row # 5	0	1	0	1	1
Row # 6	0	1	1	0	0
Row # 7	0	1	1	1	1
Row # 8	1	0	0	0	1
Row # 9	1	0	0	1	0
Row # 10	1	0	1	0	1
Row # 11	1	0	1	1	1
Row # 12	1	1	0	0	1
Row # 13	1	1	0	1	1
Row # 14	1	1	1	0	1
Row # 15	1	1	1	1	0

The corresponding K-map is:

AB / CD	00	01	11	10
00	Row # 0 0	Row # 4 1	Row # 12 1	Row # 8 1
01	Row # 1 1	Row # 5 1	Row # 13 1	Row # 9 0
11	Row # 3 1	Row # 7 1	Row # 15 0	Row # 11 1
10	Row # 2 1	Row # 6 0	Row # 14 1	Row # 10 1

Simplifying Boolean Expressions using Karnaugh map

To simplify the resulting Boolean expression using a Karnaugh map adjacent cells containing one are looped together. This step eliminated any terms of the form *AA* .

Adjacent cells means:

1. Cells that are side by side in the horizontal and vertical directions (but not diagonal).
2. For a map row: the leftmost cell and the rightmost cell.
3. For a map column: the topmost cell and the bottom most cell.
4. For a 4 variable map: cells occupying the four corners of the map.

Cells may only be looped together in twos, fours, or eights. As few groups as possible must

be formed. Groups may overlap one another and may contain only one cell.

The larger the number of 1s looped together in a group the simpler is the product term that the group represents.

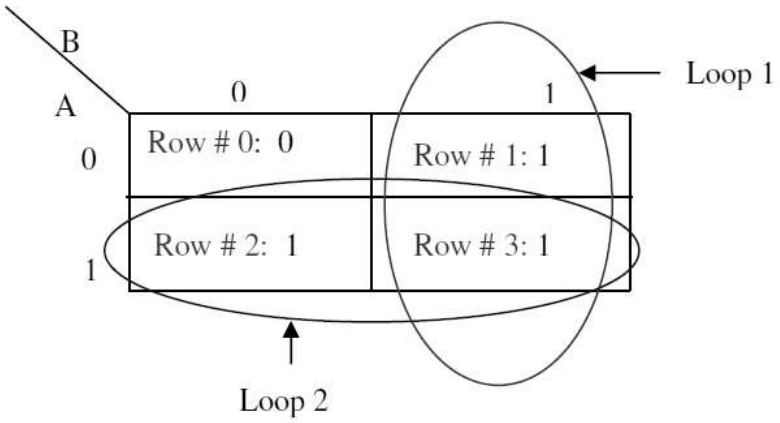
Example #1:

Simplifying the corresponding K-map of a two-input truth table for a digital circuit:

In Loop 1 the variable A has both logic 0 and logic 1 values in the same loop. B has a value of 1. Hence minterm equation is: $F = B$.

In Loop 2 Variable B have both logic 0 and 1 values in the same loop. A = 1, hence minterm equation is: *F = A.*

The overall Boolean expression for F is therefore: *F = A + B*



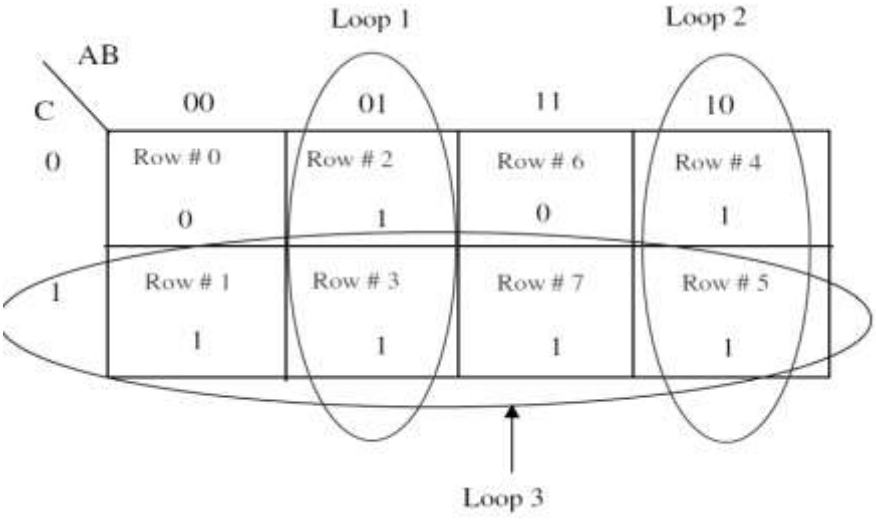
Example #2:

Simplifying the corresponding K-map of a three-input truth table for a digital circuit, In Loop 1 the variable C has both logic 0 and logic 1 values in the same loop. A has a value of 0 and B has a logic value of 1. Hence minterm equation is: *F = AB*

In Loop 2 the variable C has both logic 0 and 1 values in the same loop. A = 1 and B = 0, hence minterm equation is: *F = AB* .

In Loop 3 the two variables A and B both have logic 0 and logic 1 values in the same loop. C has a value of 1. Hence minterm equation is: *F = C.*

The overall Boolean expression for F is therefore: *F = AB + AB + C*



Example #3:

Simplifying the corresponding K-map of a four-input truth table for a digital circuit, In Loop 1 the two variables A and D both have logic 0 and logic 1 values in the same loop. C has a value of 0 and B has a value of 1. Hence minterm equation is: *F = BC.*

In Loop 2 the two variables B and C both have logic 0 and logic 1 values in the same loop. A has a value of 1 and D has a value of 0. Hence minterm equation is: *F = AD.*

In Loop 3 the variable D has logic 0 and logic 1 values in the same loop. A and B both have a value of 0 and C has a value of 1. Hence minterm equation is: *F = ABC.*

In Loop 4 the two variables B and C both have logic 0 and logic 1 values in the same loop. A has a value of 0 and D has a value of 1. Hence minterm equation is: *F = AD.*

In Loop 5 the variable C has logic 0 and logic 1 values in the same loop. A and D both have a value of 1 and B has a value of 0. Hence minterm equation is: *F = ABD.*

The overall Boolean expression for F is therefore: *F = BC + AD + ABC + AD + ABD*

COMBINATIONAL CIRCUITS

HALF ADDER

A key requirement of digital computers is the ability to use logical functions to perform arithmetic operations. The basis of this is addition; if it is possible to add two binary numbers, it is just as easily subtract them, or get a little fancier and perform multiplication and division. Then how to add two binary numbers?

Let's start by adding two binary bits. Since each bit has only two possible values, 0 or 1, there are only four possible combinations of inputs. These four possibilities, and the resulting sums, are:

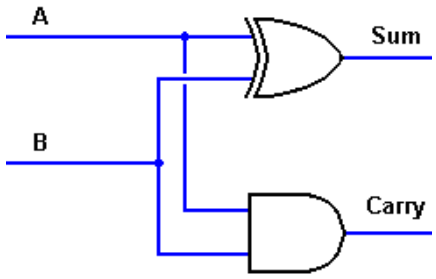
```
0 + 0 =  0
0 + 1 =  1
1 + 0 =  1
1 + 1 = 10
```

The above fourth line indicates that we have to account for two output bits when we add two input bits: the sum and a possible carry. Let's set this up as a truth table with two inputs and two outputs,

INPUTS		**OUTPUTS**	
A	**B**	**CARRY**	**SUM**

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

From the above table it is clear that, the Carry output is a simple AND function, and the Sum is an Exclusive-OR. Thus, two gates can be used to add these two bits together. The resulting circuit is shown below.



In a computer, it is very much necessary to add multi-bit numbers together. If each pair of bits can produce an output carry, it must also be able to recognize and include a carry from the next lower order of magnitude. This is the same requirement as adding decimal numbers -- if you have a carry from one column to the next; the next column has to include that carry. We have to do the same thing with binary numbers, for the same reason. As a result, the circuit to the left is known as a "half adder," because it only does half of the job. There is need a circuit that will do the entire job.

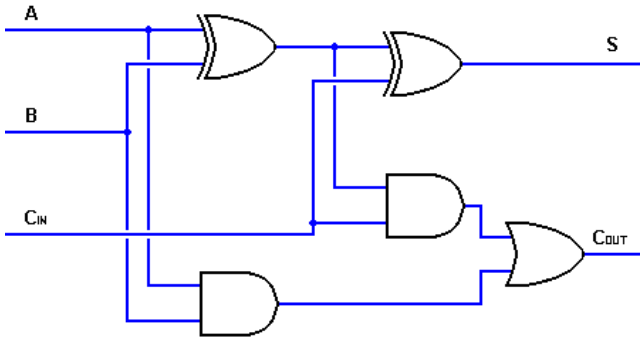
To construct a full adder circuit, we'll need three inputs and two outputs. Since we'll have both an input carry and an output carry, we'll designate them as C_{IN} and C_{OUT}. At the same time, we'll use S to designate the final Sum output. The resulting truth table is shown below.

INPUTS			**OUTPUTS**	
A	**B**	**C_{IN}**	**C_{OUT}**	**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

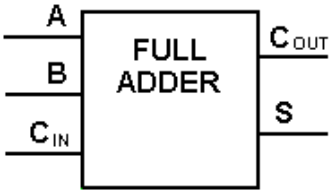
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

This is looking a bit messy. It looks as if C_{OUT} may be either an AND or an OR function, depending on the value of A, and S is either an XOR or an XNOR, again depending on the value of A. Looking a little more closely, however, we can note that the S output is actually an XOR between the A input and the half-adder SUM output with B and C_{IN} inputs. Also, the output carry will be true if any two or all three inputs are logic 1.

What this suggests is also intuitively logical: we can use two half-adder circuits. The first will add A and B to produce a partial Sum, while the second will add C_{IN} to that Sum to produce the final S output. If either half-adder produces a carry, there will be an output carry. Thus, C_{OUT} will be an OR function of the half-adder Carry outputs. The resulting full adder circuit is shown below.



The circuit above is really too complicated to be used in larger logic diagrams, so a separate symbol, shown below, is used to represent a one-bit full adder. In fact, it is common practice in logic diagrams to represent any complex function as a "black box" with input and output signals designated. It is, after all, the logical function that is important, not the exact method of performing that function.



HALF SUBTRACTOR

We have seen how simple logic gates can perform the process of binary addition. It is only logical to assume that a similar circuit could perform binary subtraction.

If we look at the possibilities involved in subtracting one 1-bit number from another, we can quickly see that three of the four possible combinations are easy and straight-forward. The fourth one involves a bit more:

0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
0 - 1 = 1, with a borrow bit.

That borrow bit is just like a borrow in decimal subtraction: it subtracts from the next higher order of magnitude in the overall number. Let's see what the truth table looks like.

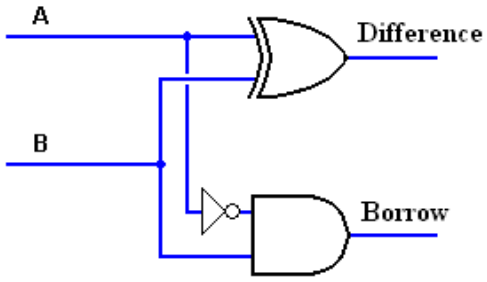
INPUTS		**OUTPUTS**	
A	**B**	**BORROW**	**A - B**
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

This is an interesting result. The difference, A-B, is still an Exclusive-OR function, just as the sum was for addition. The borrow is still an AND function, but is A'B instead of AB.

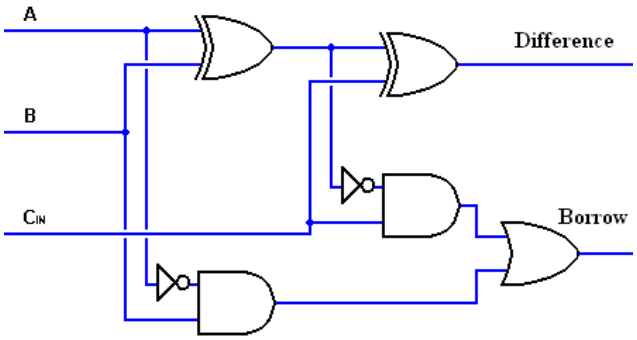
What we'd like to do, now, is find an easy way to use the binary adder to perform subtraction as well. We already have half of it working: the difference output. Can we simply invert the A input so the AND gate will have the right signals? No, we can't, because that would invert the sense of the Exclusive-OR function.

What would be really nice is to convert B to the negative equivalent of its value, and then use the basic adder just as it stands. To see if we can do that, let's consider negative binary numbers below.

The half adder circuit can be designed as designed as follows,



As like as the normal subtraction it is possible to perform the subtraction between three binary numbers. It is necessary since when multi-bit subtraction is going to be performed the borrow will be transferred to the next bit subtraction on some occasions. The full subtractor circuit is show in the below figure.



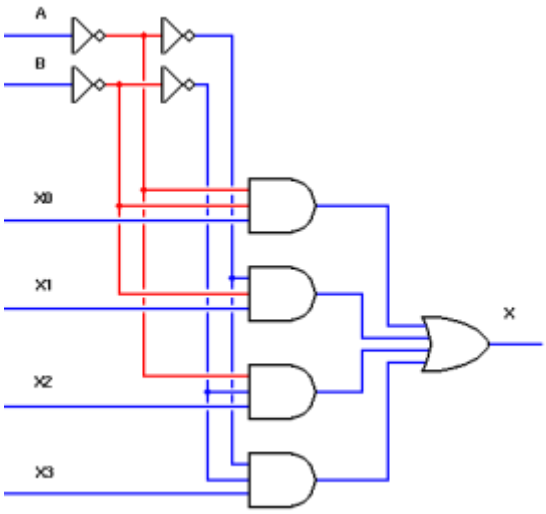
The input and output of the full subtractor is given below as a truth table,

INPUTS			**OUTPUTS**	
A	**B**	**$Borr_{IN}$**	**$Borr_{OUT}$**	**Diff**
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

MULTIPLEXER

One circuit I've received a number of requests for is the *multiplexer* circuit. This is a digital circuit with multiple signal inputs, one of which is selected by separate address inputs to be sent to the single output. It's not easy to describe without the logic diagram, but is easy to understand when the diagram is available.

The 4x1 multiplexer circuit is shown in the below figure,



The multiplexer circuit is typically used to combine two or more digital signals onto a single line, by placing them there at different times. Technically, this is known as *time-division multiplexing*.

Input A is the addressing input, which controls which of the two data inputs, X0 or X1, will be transmitted to the output. If the A input switches back and forth at a frequency more than double the frequency of either digital signal, both signals will be accurately reproduced, and can be separated again by a *Demultiplexer* circuit synchronized to the multiplexer.

This is not as difficult as it may seem at first glance; the telephone network combines multiple audio signals onto a single pair of wires using exactly this technique, and is readily able to separate many telephone conversations so that everyone's voice goes only to the intended recipient. With the growth of the Internet and the World Wide Web, most people have heard about T1 telephone lines. A T1 line can transmit up to 24 individual telephone conversations by multiplexing them in this manner.

Very common application for this type of circuit is found in computers, where dynamic memory uses the same address lines for both row and column addressing. A set of multiplexers is used to first select the row address to the memory, then switch to the column address. This scheme allows large amounts of memory to be incorporated into the computer while limiting the number of copper traces required to connect that memory to the rest of the computer circuitry. In such an application, this circuit is commonly called a *data selector*.

Multiplexers are not limited to two data inputs. If we use two addressing inputs, we can multiplex up to four data signals. With three addressing inputs, we can multiplex eight signals. If you would like to see a demonstration of a four-input multiplexer.

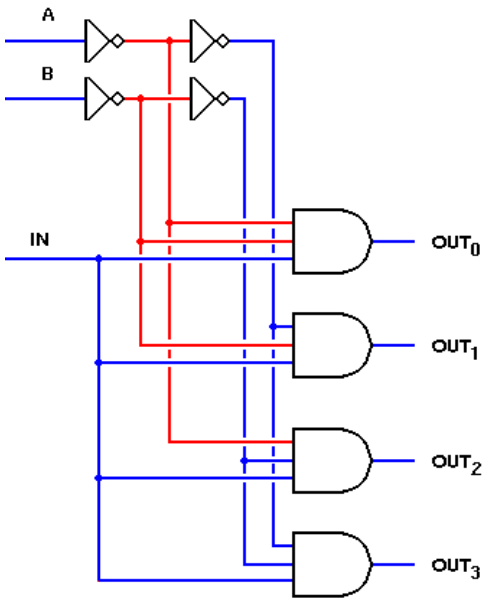
D**EMULTIPLEXER/DECODER**

The opposite of the multiplexer circuit, logically enough, is the *demultiplexer*. This circuit takes a single data input and one or more address inputs, and selects which of multiple outputs will receive the input signal. The same circuit can also be used as a *decoder*, by using the address inputs as a binary number and producing an output signal on the single output that matches the binary address input. In this application, the data input line functions as a circuit enabler — if the circuit is disabled, no output will show activity regardless of the binary input number.

This circuit uses the same AND gates and the same addressing scheme as the two-input multiplexer circuit shown in these pages. The basic difference is that it is the inputs that are combined and the outputs that are separate. By making this change, we get a circuit that is the inverse of the two-input multiplexer. If you were to construct both circuits on a single breadboard, connect the multiplexer output to the data IN of the Demultiplexer, and drive the Address inputs of both circuits with the same signal, you would find that the initial X0 input would be transmitted to OUT_0 and the X1 input would reach only OUT_1.

The one problem with this arrangement is that one of the two outputs will be inactive while the other is active. To retain the output signal, we need to add a latch circuit that can follow the data signal while it's active, but will hold the last signal state while the other data signal is active. An excellent circuit for this is the D (or Data) Latch. By placing a latch after each output and using the Addressing input (or its inverse) to control them, we can maintain both output signals at all times. If the Address input changes much more rapidly than the data inputs, the output signals will match the inputs faithfully.

A 2-to-4 line decoder/Demultiplexer is shown below.



Like the multiplexer circuit, the decoder/Demultiplexer is not limited to a single address line, and therefore can have more than two outputs. With two, three, or four addressing lines, this circuit can decode a two, three, or four-bit binary number, or can Demultiplexer up to four, eight, or sixteen time-multiplexed signals.

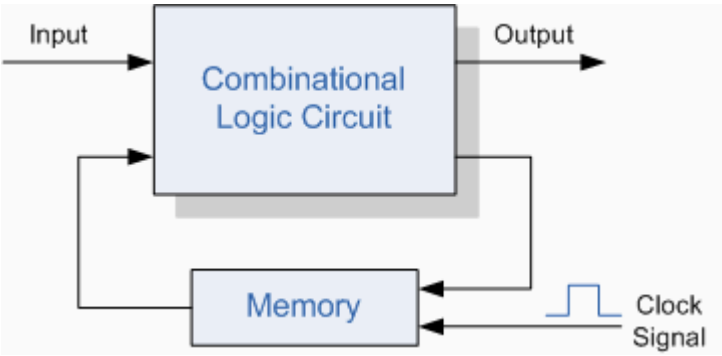
As a decoder, this circuit takes an *n*-bit binary number and produces an output on one of 2^n output lines. It is therefore commonly defined by the number of addressing input lines and the number of data output lines. Typical decoder/Demultiplexer ICs might contain two 2-to-4 line circuits, a 3-to-8 line circuit, or a 4-to-16 line circuit. One exception to the binary nature of this circuit is the 4-to-10 line decoder/Demultiplexer, which is intended to convert a BCD (Binary Coded Decimal) input to an output in the 0-9 range.

If you use this circuit as a Demultiplexer, you may want to add data latches at the outputs to retain each signal while the others are being transmitted. However, this does not apply when you are using this circuit as a decoder — then you will want only a single active output to match the input code.

Sequential Logic Basics

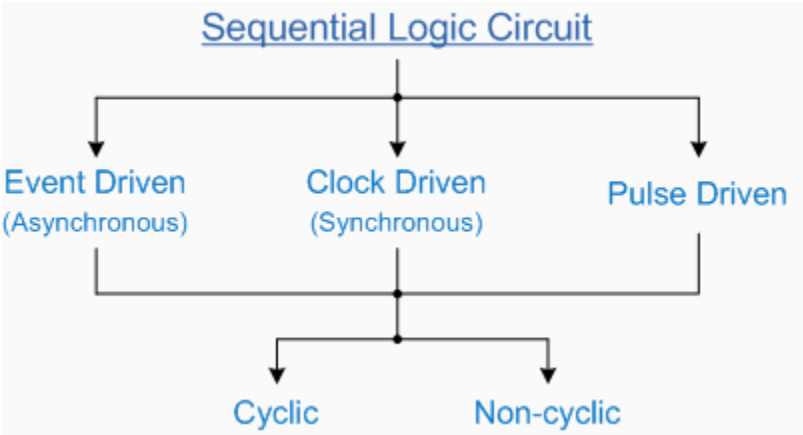
Unlike Combinational Logic circuits that change state depending upon the actual signals being applied to their inputs at that time, Sequential Logic circuits have some form of inherent "Memory" built in to them and they are able to take into account their previous input state as well as those actually present, a sort of "before" and "after" is involved. They are generally termed as Two State or Bistable devices which can have their output set in either of two basic states, a logic level "1" or a logic level "0" and will remain "latched" indefinitely in this current state or condition until some other input trigger pulse or signal is applied which will cause it to change its state once again.

Sequential Logic Circuit



The word "Sequential" means that things happen in a "sequence", one after another and in Sequential Logic circuits, the actual clock signal determines when things will happen next. Simple sequential logic circuits can be constructed from standard Bistable circuits such as Flip-flops, Latches or Counters and which themselves can be made by simply connecting together NAND Gates and/or NOR Gates in a particular combinational way to produce the required sequential circuit.

Classification of Sequential Logic



As well as the two logic states mentioned above logic level "1" and logic level "0", a third element is introduced that separates Sequential Logic circuits from their Combinational Logic counterparts, namely TIME. Sequential logic circuits that return back to their original state once reset, i.e. circuits with loops or feedback paths are said to be "Cyclic" in nature.

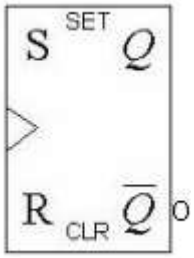
Flip-Flops

Flip-flops are synchronous bistable devices. The term synchronous means the output changes state only when the clock input is triggered. That is, changes in the output occur in synchronization with the clock.
Flip-flop is a kind of multivibrator. There are three types of multivibrators:

1. Monostable multivibrator (also called one-shot) has only one stable state. It produces a single pulse in response to a triggering input.
2. Bistable multivibrator exhibits two stable states. It is able to retain the two SET and RESET states indefinitely. It is commonly used as a basic building block for counters, registers and memories.
3. Astable multivibrator has no stable state at all. It is used primarily as an oscillator to generate periodic pulse waveforms for timing purposes.

Edge-Triggered Flip-flops

An edge-triggered flip-flop changes states either at the positive edge (rising edge) or at the negative edge (falling edge) of the clock pulse on the control input. The three basic types are introduced here: S-R, J-K and D.



The S-R, J-K and D inputs are called synchronous inputs because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse. On the other hand, the direct set (SET) and clear (CLR) inputs are called asynchronous inputs, as they are inputs that affect the state of the flip-flop independent of the clock. For the synchronous operations to work properly, these asynchronous inputs must both be kept LOW.

Edge-triggered J-K flip-flop

The J-K flip-flop works very similar to S-R flip-flop. The only difference is that this flip-flop has NO invalid state. The outputs toggle (change to the opposite state) when both J and K inputs are HIGH. The truth table is shown below.

Inputs			Outputs		
J	K	C	Q	Q'	Comments
0	0	↑	Q	Q'	No change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	Q'	Q	Toggle

Edge-triggered D flip-flop

The operations of a D flip-flop is much more simpler. It has only one input addition to the clock. It is very useful when a single data bit (0 or 1) is to be stored. If there is a HIGH on the D input

when a clock pulse is applied, the flip-flop SETs and stores a 1. If there is a LOW on the D input when a clock pulse is applied, the flip-flop RESETs and stores a 0. The truth table below summarize the operations of the positive edge-triggered D flip-flop. As before, the negative edge-triggered flip-flop works the same except that the falling edge of the clock pulse is the triggering edge.

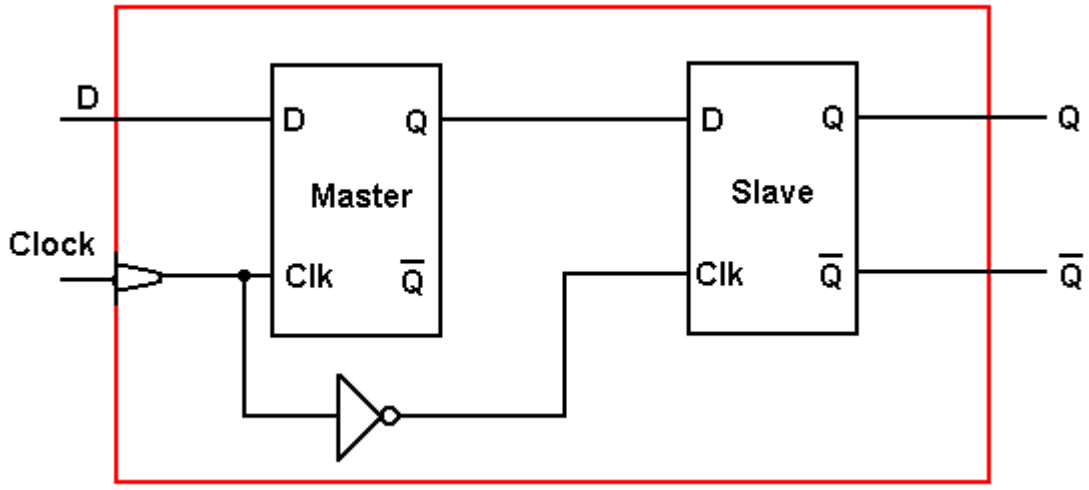
Inputs		Outputs		
D	C	Q	Q'	Comments
0	↑	0	1	RESET
1	↑	1	0	SET

Pulse-Triggered (Master-Slave) Flip-flops

The term pulse-triggered means that data are entered into the flip-flop on the rising edge of the clock pulse, but the output does not reflect the input state until the falling edge of the clock pulse. As this kind of flip-flops are sensitive to any change of the input levels during the clock pulse is still HIGH, the inputs must be set up prior to the clock pulse's rising edge and must not be changed before the falling edge. Otherwise, ambiguous results will happen.

The three basic types of pulse-triggered flip-flops are S-R, J-K and D. Their logic symbols are shown below. Notice that they do not have the dynamic input indicator at the clock input but have postponed output symbols at the outputs.

The truth tables for the above pulse-triggered flip-flops are all the same as that for the edge-triggered flip-flops, except for the way they are clocked. These flip-flops are also called Master-Slave flip-flops simply because their internal construction are divided into two sections. The slave section is basically the same as the master section except that it is clocked on the inverted clock pulse and is controlled by the outputs of the master section rather than by the external inputs. The logic diagram for a basic master-slave S-R flip-flop is shown below.



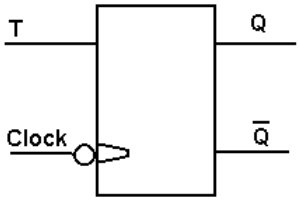
The master/slave flip-flop overcomes the following problems.

- ✓ RIPPLE THROUGH. An input changes level during the clock period, and the change appears at the output.
- ✓ PROPAGATION DELAY. The time between applying a signal to an input, and the resulting change in the output.

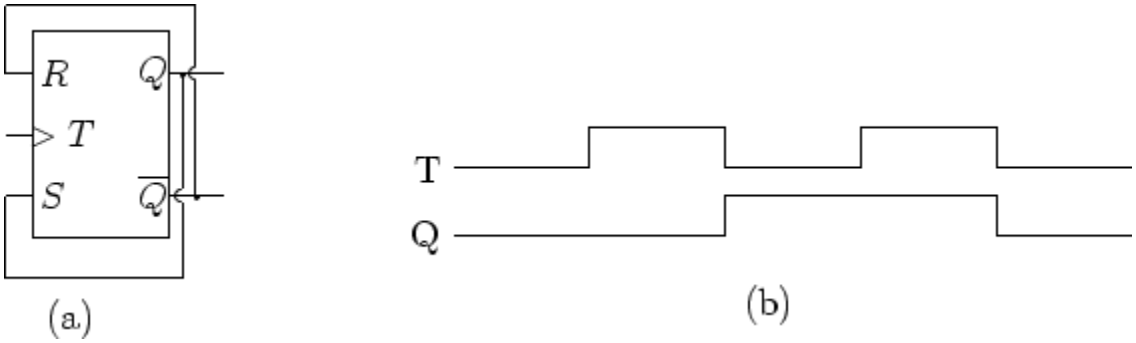
These can give problems in logic circuits. The maste/slave flipflop consists of two rising edge triggered D type flip-flops. The clock of the slave is fed via an inverter so that the falling edge of the origonal clock pulse becomes a rising edge. The slave clock pulse is an inverted version of the clock pulse shown in the lower diagram. The flip-flops are triggered at different levels of the clock pulse edge. When data is to be entered, the slave is isolated from the master, so that changes at the input do not appear at the output.Data on D is passed to Q of the master. The master is then isolated from the D input. Data, from the Q of the master, is passed to Q of the slave.

t1. Slave isolated from Master.
t2. Master connected to D input.
t3. Master isolated from D input.
t4. Master Q connected Slave D.

Toggle Flip-Flop (T)



This flip-flop toggles (Q changes state) on the negative going edge of the clock pulse. T acts as an ENABLE / INHIBIT control. Q will only toggle on the negative edge of the clock pulse, when T is high. Below is shown a D type flip-flop connected as a toggle type. On each clock pulse positive going edge, Q will go to the state bar Q was before the clock pulse arrived. Remember that bar Q is the opposite level to Q. Therefore Q will toggle. This type of flip-flop is a simplified version of the JK flip-flop. It is not usually found as an IC chip by itself, but is used in many kinds of circuits, especially counter and dividers. Its only function is that it toggles itself with every clock pulse (on either the leading edge, on the trailing edge) it can be constructed from the RS flip-flop as shown in Figure (a).



This flip flop is normally set, or ``loaded" with the preset and clear inputs. It can be used to obtain an output pulse train with a frequency of half that of the clock pulse train, as seen from the timing diagram, Figure (b). In this example, the T flip flop is triggered on the falling edge of the clock pulse. Several T flip-flops are often connected together in a simple IC to form a ``divide by N" counter, where N is usually 5, 10, 12 or a power of 2.

Overall Behavior

Each flip-flop stores a single bit of data, which is emitted through the *Q* output on the east side. Normally, the value can be controlled via the inputs to the west side. In particular, the value changes when the **clock** input, marked by a triangle on each flip-flop, rises from 0 to 1; on this rising edge, the value changes according to the corresponding table below.

D Flip-Flop T Flip-Flop J-K Flip-Flop S-R Flip-Flop

D Q	***T Q***	***J K Q***	***S R Q***
0 0	0 *Q*	0 0 *Q*	0 0 *Q*
1 1	1 *Q*'	0 1 0	0 1 0
		1 0 1	1 0 1
		1 1 *Q*'	1 1 *??*

Another way of describing the different behavior of the flip-flops is as follows:

- **D Flip-Flop:** When the clock rises from 0 to 1, the value remembered by the flip-flop becomes the value of the *D* input (*Data*) at that instant.
- **T Flip-Flop:** When the clock rises from 0 to 1, the value remembered by the flip-flop either toggles or remains the same depending on whether the *T* input (*Toggle*) is 1 or 0.
- **J-K Flip-Flop:** When the clock rises from 0 to 1, the value remembered by the flip-flop toggles if the *J* and *K* inputs are both 1, remains the same if they are both 0, and changes to the *K* input value if *J* and *K* are not equal. (The names *J* and *K* do not stand for anything.)
- **R-S Flip-Flop:** When the clock rises from 0 to 1, the value remembered by the flip-flop remains unchanged if *R* and *S* are both 0, becomes 0 if the *R* input (*Reset*) is 1, and becomes 1 if the *S* input (*Set*) is 1. The behavior in unspecified if both inputs are 1. (In Logisim, the value in the flip-flop remains unchanged.)

Shift Registers

Shift Registers consists of a number of single bit "D-Type Data Latches" connected together in a chain arrangement so that the output from one data latch becomes the input of the next latch and so on, thereby moving the stored data serially from either the left or the right direction. The number of individual Data Latches used to make up Shift Registers are determined by the number of bits to be stored. The most common used is 8-bits wide. Shift Registers are mainly used to store data and to convert data from either a serial to parallel or parallel to serial format with all the latches being driven by a common clock (Clk) signal making them Synchronous devices. They are generally provided with a Clear or Reset connection so that they can be "SET" or "RESET" as required.

Generally, Shift Registers operate in one of four different modes:

- Serial-in to Parallel-out (SIPO)
- Serial-in to Serial-out (SISO)
- Parallel-in to Parallel-out (PIPO)
- Parallel-in to Serial-out (PISO)

Serial-in to Parallel-out.

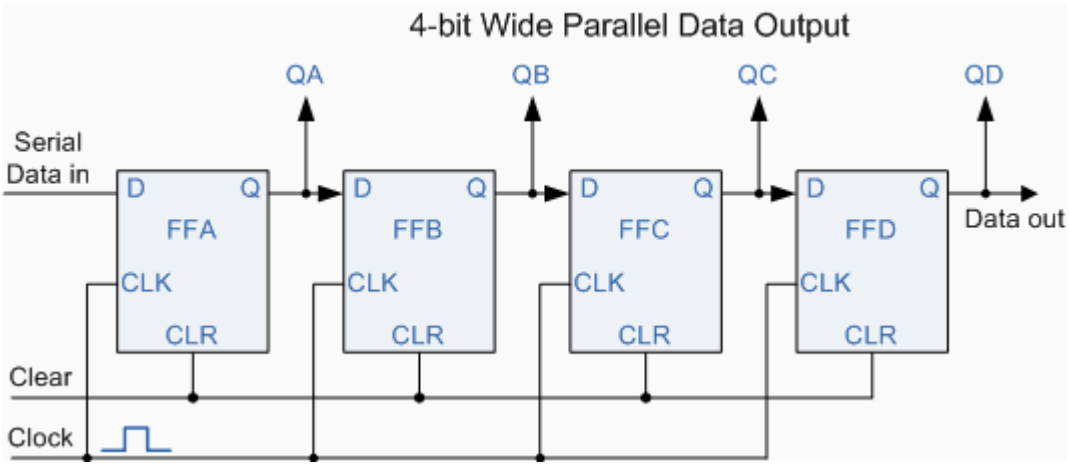


Fig: 4-bit Serial-in to Parallel-out (SIPO) Shift Register

Lets assume that all the flip-flops (FFA to FFD) have just been RESET (CLEAR input) and that all the outputs QA to QD are at logic level "0" ie, no parallel data output. If a logic "1" is connected to the DATA input pin of FFA then on the first clock pulse the output of FFA and the resulting QA will be set HIGH to logic "1" with all the other outputs remaining LOW at logic "0". Assume now that the DATA input pin of FFA has returned LOW to logic "0". The next clock pulse will change the output of FFA to logic "0" and the output of FFB and QB HIGH to logic "1". The logic "1" has now moved or been "Shifted" one place along the register to the right. When the third clock pulse arrives this logic "1" value moves to the output of FFC (QC) and so on until the arrival of the fifth clock pulse which sets all the outputs QA to QD back again to logic level "0" because the input has remained at a constant logic level "0".

The effect of each clock pulse is to shift the DATA contents of each stage one place to the right, and this is shown in the following table until the complete DATA is stored, which can now be read directly from the outputs of QA to QD. Then the DATA has been converted from a Serial Data signal to a Parallel Data word.

Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

Serial-in to Serial-out

This Shift Register is very similar to the one above except where as the data was read directly in a parallel form from the outputs QA to QD, this time the DATA is allowed to flow straight through the register. Since there is only one output the DATA leaves the shift register one bit at a time in a serial pattern and hence the name Serial-in to Serial-Out Shift Register.

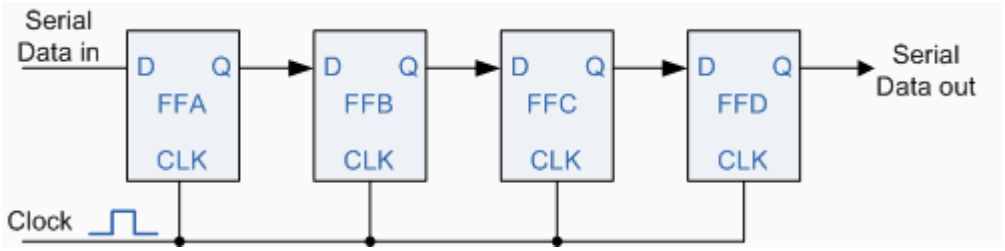


Fig: 4-bit Serial-in to Serial-out (SISO) Shift Register

This type of Shift Register also acts as a temporary storage device or as a time delay device, with the amount of time delay being controlled by the number of stages in the register, 4, 8, 16 etc or by varying the application of the clock pulses. Commonly available IC's include the 74HC595 8-bit Serial-in/Serial-out Shift Register with 3-state outputs.

Parallel-in to Serial-out

Parallel-in to Serial-out Shift Registers act in the opposite way to the Serial-in to Parallel-out one above. The DATA is applied in parallel form to the parallel input pins PA to PD of the register and is then read out sequentially from the register one bit at a time from PA to PD on each clock cycle in a serial format.

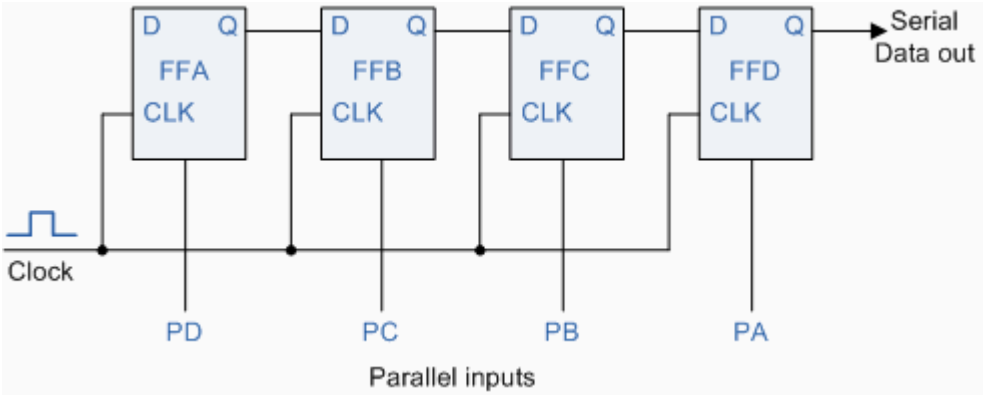


Fig: 4-bit Parallel-in to Serial-out (PISO) Shift Register

As this type of Shift Register converts parallel data, such as an 8-bit data word into serial data it can be used to multiplex many different input lines into a single serial DATA stream which can be sent directly to a computer or transmitted over a communications line. Commonly available IC's include the 74HC165 8-bit Parallel-in/Serial-out Shift Registers.

Parallel-in to Parallel-out

Parallel-in to Parallel-out Shift Registers also act as a temporary storage device or as a time delay device. The DATA is presented in a parallel format to the parallel input pins PA to PD and then shifts it to the corresponding output pins QA to QD when the registers are clocked.

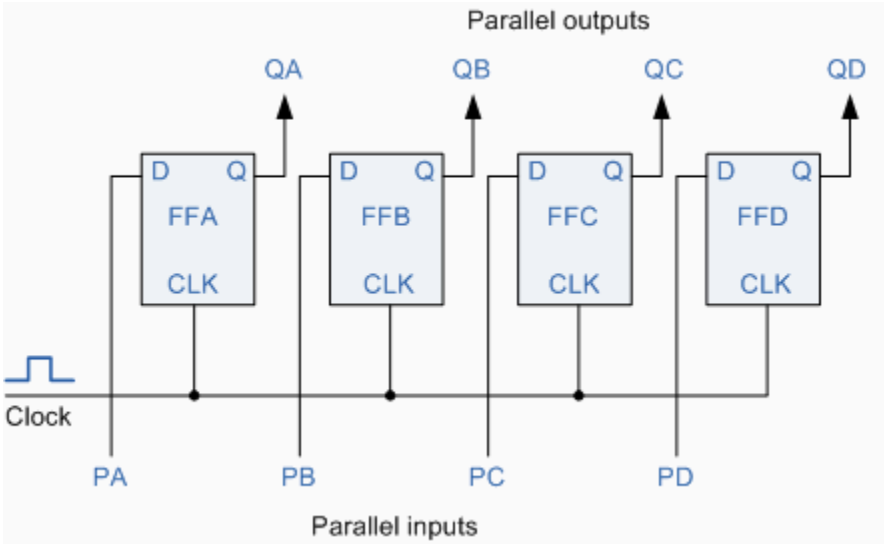


Fig : 4-bit Parallel-in/Parallel-out (PIPO) Shift Register

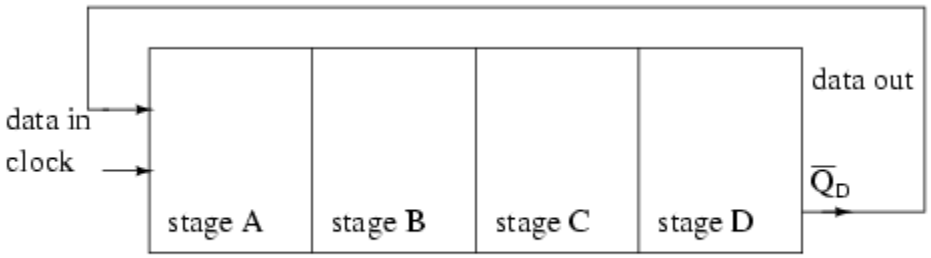
As with the Serial-in to Serial-out shift register, this type of register also acts as a temporary storage device or as a time delay device, with the amount of time delay being varied by the frequency of the clock pulses.

Today, high speed bi-directional universal type Shift Registers such as the TTL 74LS194, 74LS195 or the CMOS 4035 are available as a 4-bit multi-function devices that can be used in serial-serial, shift left, shift right, serial-parallel, parallel-serial, and as a parallel-parallel Data Registers, hence the name "Universal".

COUNTERS

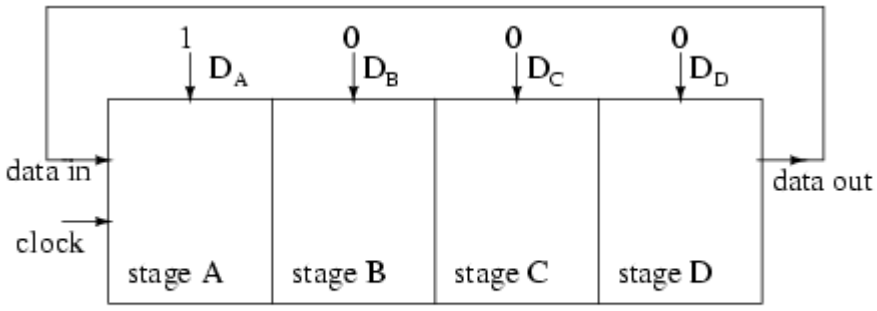
Ring counters

If the output of a shift register is fed back to the input. a ring counter results. The data pattern contained within the shift register will recirculate as long as clock pulses are applied. For example, the data pattern will repeat every four clock pulses in the figure below. However, we must load a data pattern. All 0's or all 1's doesn't count. Is a continuous logic level from such a condition useful?



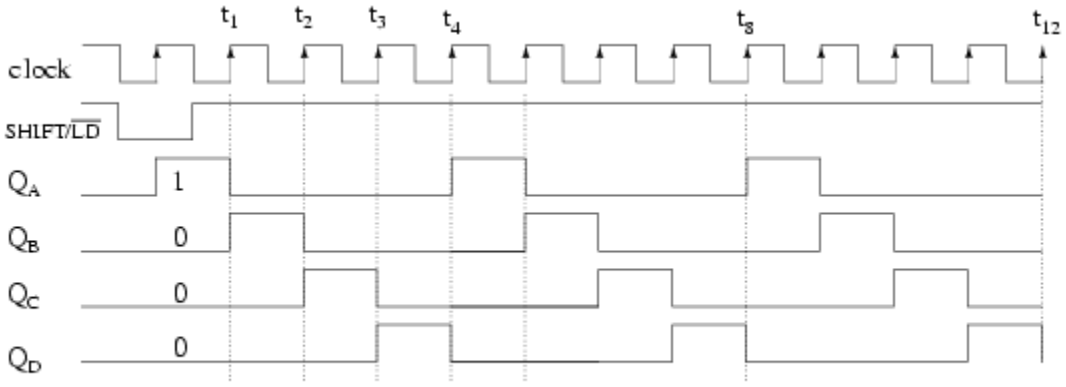
Ring Counter, shift register output fed back to input

We make provisions for loading data into the parallel-in/ serial-out shift register configured as a ring counter below. Any random pattern may be loaded. The most generally useful pattern is a single 1.



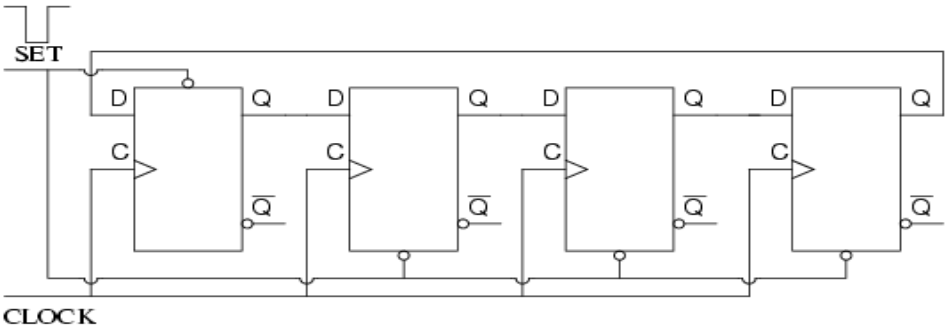
Parallel-in, serial-out shift register configured as a ring counter

Loading binary 1000 into the ring counter, above, prior to shifting yields a viewable pattern. The data pattern for a single stage repeats every four clock pulses in our 4-stage example. The waveforms for all four stages look the same, except for the one clock time delay from one stage to the next. See figure below.



Load 1000 into 4-stage ring counter and shift

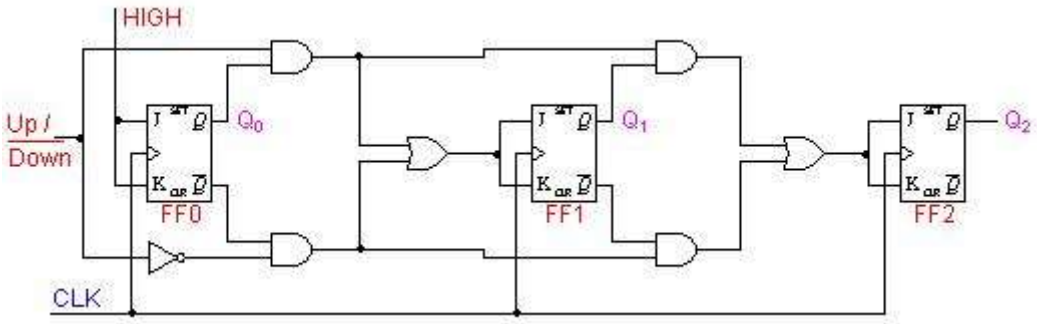
The circuit above is a divide by 4 counter. Comparing the clock input to any one of the outputs, shows a frequency ratio of 4:1. How may stages would we need for a divide by 10 ring counter? Ten stages would recirculate the 1 every 10 clock pulses.



Set one stage, clear three stages

Up-Down Counters

A circuit of a 3-bit synchronous up-down counter and a table of its sequence are shown below. Similar to an asynchronous up-down counter, a synchronous up-down counter also has an up-down control input. It is used to control the direction of the counter through a certain sequence.



An examination of the sequence table shows:

Up	Q2	Q1	Q0	Down
	0	0	0	
	0	0	1	
	0	1	0	
	0	1	1	
	1	0	0	
	1	0	1	
	1	1	0	
	1	1	1	

- for both the UP and DOWN sequences, Q0 toggles on each clock pulse.
- for the UP sequence, Q1 changes state on the next clock pulse when Q0=1.
- for the DOWN sequence, Q1 changes state on the next clock pulse when Q0=0.
- for the UP sequence, Q2 changes state on the next clock pulse when Q0=Q1=1.
- for the DOWN sequence, Q2 changes state on the next clock pulse when Q0=Q1=0.

TEXT BOOKS:

1. Carl Hamacher ,(2012). Computer Organization,(5th ed.). McGraw Hill
2. Dos Reis,A.J.,(2009), Assembly Language and Computer Architecture using C++ and JAVA Technology.
3. M.Mories Mano(2013) Computer Systems Architecture,(3rd ed.), Prentice Hall of India
4. Stalings, W., (2010). Computer Organization and Architecture Designing for Performance,(8th ed.), Prentice Hall of India

PART –A(One Mark Questions- Online Examination)

1. Flip-Flop is a ____ bit register.
   a. 1 b.3 c.2 d.4
2. Combinational circuits output depends the ____
   a. present input b. present and past input c. past output d.future input
3. Boolean Algebra A'+ A=____
   a. 0 b. 1 c. A d. A'
4. Sequential circuits output depends only the
   a. present input b. present and past input c. past output d.future input
5. What is the base number of hexadecimal number
   a. 2 b. 4 c. 8 d.16
6. What is the one's complement of $(1011110)_2$
   a. 0100001 b. 0111101 c. 1010000 d. 0001110
7. Convert $(255)_{10}$ to binary number ( )$_2$
   a. 111111110 b.100101010 c. 101010101 d. 010101010
8. Signed number is used to indicate ____ numbers
   a. negative b. positive c. negative and positive d. zero
9. When the CPU detects an interrupt, it then saves its
   a. previous state b. next state c. current state d. future state
10. A microprogram is sequencer perform the operation____
    a. read b. write c. read and write d. read and execute
11. The memory unit has a capacity of _____ words.
    a. 128 b. 4096 c.64 d. 256

PART – B(2 Marks Questions)

1. Define Flip-Flop.
2. Convert the binary number $(1011101)_2$ into decimal number.
3. Define combinational circuits.
4. Convert the binary number $(1011101010000010101)_2$ into octal number
5. Define Multiprocessing.
6. What is a Co-Processor?

PART - C (6 Marks Questions)

1. Discuss in detail Boolean properties with an example.
2. Explain in detail 4:1 multiplexer with neat diagram.
3. Discuss in detail shift registers with an example.
4. Explain in detail 8:1 multiplexer with neat diagram.
5. Simplify the Boolean function using K-map.
   F(A,B,C,D,E) = (0,2,4,6,9,13,14,15)
6. What is a half-adder? Explain a half-adder with the help of truth-table and logic diagram.
7. What are sequential Circuits? Explain encoder and decoder.
8. Simplify the Boolean expressions: i) $Z = f(A,B) = A\overline{B} + AB$
   ii) $Z = f(A,B) = \overline{A}\overline{B} + A\overline{B} + \overline{A}B$ iii) $Z = f(A,B,C) = \overline{A}\overline{B}\,\overline{C} + \overline{A}B + AB\overline{C} + AC$
   iv) $Z = f(A,B,C) = \overline{A}B + B\overline{C} + BC + A\overline{B}\overline{C}$

UINT – II

Numbering System

Many number systems are in use in digital technology. The most common are the decimal, binary, octal, and hexadecimal systems. The decimal system is clearly the most familiar to us because it is a tool that we use every day. Examining some of its characteristics will help us to better understand the other systems. The four numerical representation systems that are used most commonly in the digital system are,

- Decimal
- Binary
- Octal
- Hexadecimal

Decimal System

The decimal system is composed of 10 numerals or symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Using these symbols as digits of a number, we can express any quantity. The decimal system is also called the base-10 system because it has 10 digits.

10^3	10^2	10^1	10^0
=1000	=100	=10	=1

Most Significant Digit

Even though the decimal system has only 10 symbols, any number of any magnitude can be expressed by using our system of positional weighting.

Decimal Examples

- 52_{10}
- 1024_{10}
- 64000_{10}

Binary System

In the binary system, there are only two symbols or possible digit values, 0 and 1. This base-2 system can be used to represent any quantity that can be represented in decimal or other base system. This system has been developed since the machines can understand only two logics, 1 or 0.

2^3	2^2	2^1	2^0
=8	=4	=2	=1

Most Significant Digit

Binary Examples

- 11_2
- 1010_2
- 11000_2
-

Binary Counting

The Binary counting sequence is shown in the table:

2^3	2^2	2^1	2^0	Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Representing Binary Quantities

In digital systems the information that is being processed is usually presented in binary form. Binary quantities can be represented by any device that has only two operating states or possible conditions. E.g. a switch is only open or closed. We arbitrarily (as we define them) let an open switch represent binary 0 and a closed switch represent binary 1. Thus we can represent any binary number by using series of switches.

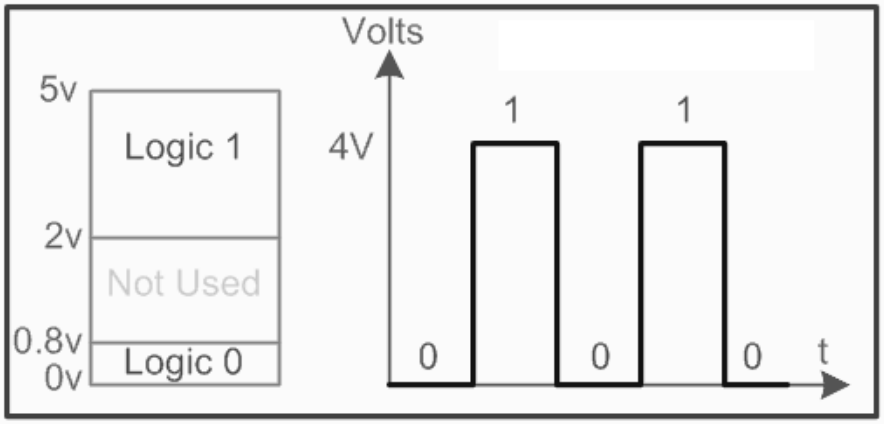
Typical Voltage Assignment

Binary 1: Any voltage between 2V to 5V
Binary 0: Any voltage between 0V to 0.8V

Not used: Voltage between 0.8V to 2V in 5 Volt CMOS and TTL Logic, this may cause error in a digital circuit. Today's digital circuits works at 1.8 volts, so this statement may not hold true for all logic circuits.

We can see another significant difference between digital and analog systems. In digital systems, the exact voltage value is not important; eg, a voltage of 3.6V means the same as a voltage of 4.3V. In analog systems, the exact voltage value is important.

The binary number system is the most important one in digital systems, but several others are also important. The decimal system is important because it is universally used to represent quantities outside a digital system. This means that there will be situations where decimal values have to be converted to binary values before they are entered into the digital system.



In additional to binary and decimal, two other number systems find wide-spread applications in digital systems. The octal (base-8) and hexadecimal (base-16) number systems are both used for the same purpose- to provide an efficient means for representing large binary system.

Octal System

The octal number system has a base of eight, meaning that it has eight possible digits: 0, 1, 2, 3, 4, 5, 6, and 7.

8^3	8^2	8^1	8^0
=512	=64	=8	=1

Most Significant Digit

Octal Examples

- 237_8
- 24_8
- 11_8

Hexadecimal System

The hexadecimal system uses base 16. Thus, it has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters A, B, C, D, E, and F as the 16 digit symbols.

16^3	16^2	16^1	16^0
=4096	=256	=16	=1
Most Significant Digit			

Hexadecimal Examples

- 24_{16}
- 11_{16}

Conversion:

Converting from one code form to another code form is called code conversion, like converting from binary to decimal or converting from hexadecimal to decimal.

Binary-To-Decimal Conversion

Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number which contain a 1.

Binary	**Decimal**
11011_2	
$2^4(1)+2^3(1)+2^2(0)+2^1(1)+2^0(1)$	=16+8+0+2+1
Result	27_{10}

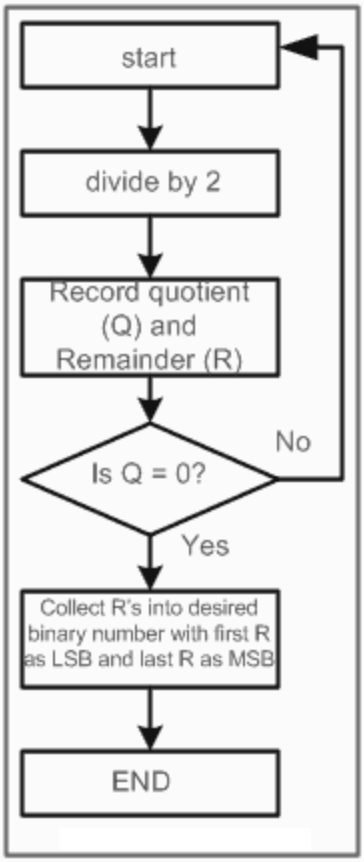
You should have noticed that the method is to find the weights (i.e., powers of 2) for each bit position that contains a 1, and then to add them up.

Decimal-To-Binary Conversion

The method of converting the decimal number to binary is called as Double dabble (Repeated division by 2) method. Convert 25_{10} to binary

Division	**Remainder**	**Binary**
25/2	= 12+ remainder of 1	1 (Least Significant Bit)
12/2	= 6 + remainder of 0	0
6/2	= 3 + remainder of 0	0
3/2	= 1 + remainder of 1	1
1/2	= 0 + remainder of 1	1 (Most Significant Bit)
Result	25_{10}	= 11001_2

The Flow chart for Double dabble method is as follows:



Octal-To-Decimal Conversion

Any octal number can be converted to its decimal equivalent simply by summing together the product of weights (power of 8) of the various positions and the number present in that position in the octal number.

Example:

$236_{16} = 2 \times (8^2) + 3 \times (8^1) + 6 \times (8^0) = 158_{10}$

Decimal-To-Octal Conversion

The method of converting the decimal number to octal is called as Octal dabble (Repeated division by 8) method.

Example: Convert 177_{10} to octal and binary

Division	**Result**	**Binary**
177/8	= 22+ remainder of 1	1 (Least Significant Bit)
22/ 8	= 2 + remainder of 6	6
2 / 8	= 0 + remainder of 2	2 (Most Significant Bit)
Result	177_{10}	$= 261_8$
Binary		$= 010110001_2$

Hexadecimal-To-Decimal Conversion

Any hexadecimal number can be converted to its decimal equivalent simply by summing together the product of weights (power of 16) of the various positions and the number present in that position in the hexadecimal number number.

Example:

$2AF_{16} = 2 \times (16^2) + 10 \times (16^1) + 15 \times (16^0) = 687_{10}$

Decimal-To-Hexadecimal Conversion

The method of converting the decimal number to hexadecimal is called as Hex dabble (Repeated division by 16) method

Example: convert 378_{10} to hexadecimal and binary:

Division	**Result**	**Hexadecimal**
378/16	= 23+ remainder of 10	A (Least Significant Bit)23
23/16	= 1 + remainder of 7	7
1/16	= 0 + remainder of 1	1 (Most Significant Bit)
Result	378_{10}	$= 17A_{16}$
Binary		$= 0001\ 0111\ 1010_2$

Binary-To-Octal / Octal-To-Binary Conversion

Octal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	000	001	010	011	100	101	110	111

Each Octal digit is represented by three binary digits.

Example:

$100\ 111\ 010_2 = (100)\ (111)\ (010)_2 = 4\ 7\ 2_8$

Binary-To-Hexadecimal /Hexadecimal-To-Binary Conversion

Hexadecimal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal Digit	8	9	A	B	C	D	E	F
Binary Equivalent	1000	1001	1010	1011	1100	1101	1110	1111

Each Hexadecimal digit is represented by four bits of binary digit**.**

Example:

$1011\ 0010\ 1111_2 = (1011)\ (0010)\ (1111)_2 = B\ 2\ F_{16}$

Floating Point Numbers

A real number or floating point number is a number which has both an integer and a fractional part.
Examples for real decimal numbers are 123.45_{10}, $0.1234_{10,}$ etc.

10^3	10^2 10^1 10^0		10^{-1} 10^{-2}	10^{-3}
=1000	=100 =10 =1	.	=0.1 =0.01	=0.001
Most Significant Digit		Decimal point		Least Significant Digit

Examples for real binary numbers are 1100.1100_2, 0.1001_2, etc.

2^3	2^2 2^1 2^0		2^{-1} 2^{-2}	2^{-3}
=8	=4 =2 =1	.	=0.5 =0.25	=0.125
Most Significant Digit		Binary point		Least Significant Digit

Examples for real octal numbers are $12.3_{8,}$ $11.1_{8,}$ etc.

8^3	8^2 8^1 8^0		8^{-1} 8^{-2}	8^{-3}
=512	=64 =8 =1	.	=1/8 =1/64	=1/512
Most Significant Digit		Octal point		Least Significant Digit

Examples for real hexadecimal numbers are 24.6_{16}, $11.1_{16,}$ etc.

16^3	16^2 16^1 16^0		16^{-1} 16^{-2}	16^{-3}
=4096	=256 =16 =1	.	=1/16 =1/256	=1/4096
Most Significant Digit		Hexa Decimal point		Least Significant Digit

Binary Addition

The binary addition is very much similar to decimal addition and the addition of two bits is given as,

```
0 + 0 = 00      ; Sum 0 and Carry 0
0 +1 = 01       ; Sum 1 and Carry 0
1 + 0 = 01      ; Sum 1 and Carry 0
1 + 1 = 10      ; Sum 0 and Carry 1
```

In the above addition the result is separated into two parts such as Sum and Carry.

When the addition of multiple bits is performed the carry of previous bit will be added with current bit addition. When the carry is zero the above addition result is followed. When the carry is one the result will be as follows,

```
1 + 0 + 0 = 00 ; Sum 1 and Carry 0
1 + 0 +1 = 01  ; Sum 0 and Carry 1
1 + 1 + 0 = 01 ; Sum 0 and Carry 1
1 + 1 + 1 = 10 ; Sum 1 and Carry 1
```

Example:

```
  011        3
+100       +4
  111        7
```

Binary Subtraction

The binary subtraction very much similar to decimal subtraction the subtraction two bits is given as,

```
0 - 0 = 00      ; Difference 0 and Borrow 0
0 - 1 = 11      ; Difference 1 and Borrow 1
1 - 0 = 10      ; Difference 1 and Borrow 0
1 - 1 = 00      ; Difference 0 and Borrow 0
```

In the above addition the result is separated into two parts such as Difference and Borrow.

When the Subtraction of multiple bits is performed, the borrow of previous bit will be subtracted with current bit subtraction. When, the borrow is zero the above subtraction result is followed. When the borrow is one the result will be as follows,

```
1 + 0 + 0 = 11 ; Difference 1 and Borrow 1
1 + 0 +1 = 01  ; Difference 0 and Borrow 1
1 + 1 + 0 = 00 ; Difference 0 and Borrow 0
1 + 1 + 1 = 11 ; Difference 1 and Borrow 1
```

Example:

```
  110          6
- 101         -5
  001          1
```

Note: If the subtrahend is grater than the minuend, then the result will be represented indirectly (2's complement of the difference).

Binary Multiplication

The multiplication of two bits is given as,

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

The multiplication can be done by two different methods, one is by using the above rules as like normal multiplication, another by repeated addition method. In repeated addition method the multiplicand will be added with the same value and the count (multiplier) value is decremented by one and this process will be continued until the count (multiplier) value reaches zero.

Example:

Normal method:

```
  111 x   multiplicand        7 x
  101     multiplier          5
  111                         35
 000
111
100011
```

Repeated addition method:

```
                  111    (multiplicand) count = 101 (multiplier) – 1
if, count > 0     +111                  count = count -1
                  1110
if, count > 0     +111                  count = count -1
                   ...
```

it continues until the count reaches zero.

Binary Division

The division of two bits is given as,

0 / 0 = 0
1 / 1 = 1

The division can be done by two different methods, one is by using the above rules as like normal division, another by repeated subtraction method. In repeated subtraction method the divisor value will be subtracted from the dividend and the count (quotient) value is incremented by one and this process will be continued until the dividend value is greater than divisor value. the final value of the dividend that is less than the divisor value is remainder.

Example:

Normal method:

```
     10                      3
11 | 110   dividend     3 |  6
     11    multiplier        3
     00                      0
```

Repeated subtraction method:

```
                      110    (dividend)    count = 0 (quotient)
if, dividend > divisor -011                count = count +1 (i.e., 01)
                       011
if, dividend > divisor -011                count = count +1 (i.e., 10)
                       00
```

Stop the subtraction process since the dividend value (00) is less than the divisor value (11). The final dividend value "00" is remainder and the final count value "10" is the quotient.

1's and 2's Complement

There are two types of complement for binary number system, namely, 1's complement and 2's complement. 1's complement and 2's complement can be used to perform subtraction using adder. Also they are used to represent negative numbers.

1's complement

The 1's complement of the binary digit (bit) is defined as 1 minus that bit,

i.e., 1's complement of 1 is 1 - 1 = 0
1's complement of 0 is 1 - 0 = 1

Example:

Binary number	1's cimplement
1011	0100
11010	00101

2's complement

The 2's complement of the binary digit (bit) is adding 1 with the 1's complement value of that number.

Example:

Binary number	2's complement
1001	0111
10010	01110

Subtraction using complement method

1's complement subtraction

In this method the following three steps are followed,

- Take the minuend in binary format as it is.
- Take the 1's complement for subtrahend and add with the minuend.
- If carry is produced, add the carry to the sum.

The carry produced is called end-around carry. If carry is not produced it indicates that the result is negative and the result is in 1's complement method.

Example:

110 (minuend) 6
010 (subtrahend) -5
1 000 1

If the carry is '1', then add the carry with the result and the result is positive. The final answer is 001.

101 (minuend) 5
001 (subtrahend) -6
0 110 -1

If the carry is '0', the result is negative and the 1's complement of the result will give the difference. The final answer is 001.

2's complement subtraction

In this method the following steps are followed,

- Take the minuend in binary format as it is.
- Take the 2's complement for subtrahend and add it with the minuend.
- If carry is produced, ignore the carry.

Example:

110 (minuend)	6
011 (subtrahend)	-5
1 001	1

If the carry is '1', then the result is positive and the direct answer will be produced. The final answer is 001.

101 (minuend)	5
010 (subtrahend)	-6
0 111	-1

If the carry is '0', the result is negative and the 2's complement of the result will give the difference. The final answer is 001.

Binary Codes

Binary codes are codes which are represented in binary system with modification from the original ones. The different types of codes are:

- Weighted codes
- Non Weighted Codes

Weighted Codes

Weighted binary codes are those which obey the positional weighting principles, each position of the number represents a specific weight. The binary counting sequence is an example.

Decimal	8421	2421	5211	Excess-3
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0010	0011	0101
3	0011	0011	0101	0110
4	0100	0100	0111	0111
5	0101	1011	1000	1000
6	0110	1100	1010	1001
7	0111	1101	1100	1010
8	1000	1110	1110	1011
9	1001	1111	1111	1100

8421 Code/BCD Code

The BCD (Binary Coded Decimal) is a straight assignment of the binary equivalent. It is possible to assign weights to the binary bits according to their positions. The weights in the BCD code are 8,4,2,1.

Example: The bit assignment 1001 can be seen by its weights to represent the decimal 9 because:

1x8+0x4+0x2+1x1 = 9

2421 Code

This is a weighted code, its weights are 2, 4, 2 and 1. A decimal number is represented in 4-bit form and the total four bits weight is 2 + 4 + 2 + 1 = 9. Hence the 2421 code represents the decimal numbers from 0 to 9.

5211 Code

This is a weighted code, its weights are 5, 2, 1 and 1. A decimal number is represented in 4-bit form and the total four bits weight is 5 + 2 + 1 + 1 = 9. Hence the 5211 code represents the decimal numbers from 0 to 9

Non Weighted Codes

Non weighted codes are codes that are not positionally weighted. That is, each position within the binary number is not assigned a fixed value.

Excess-3 Code

Excess-3 is a non weighted code used to express decimal numbers. The code derives its name from the fact that each binary code is the corresponding 8421 code plus 0011(3).

Example: 1000 of 8421 = 1011 in Excess-3

Gray Code

The gray code belongs to a class of codes called minimum change codes, in which only one bit in the code changes when moving from one code to the next. The Gray code is non-weighted code, as the position of bit does not contain any weight. The gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit-distance code. In digital Gray code has got a special place.

Decimal Number	Binary Code	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Binary to Gray Conversion

- Gray Code MSB is binary code MSB.
- Gray Code MSB-1 is the XOR of binary code MSB and MSB-1.
- MSB-2 bit of gray code is XOR of MSB-1 and MSB-2 bit of binary code.

MSB-N bit of gray code is XOR of MSB-N-1 and MSB-N bit of binary code.

Error Detecting and Correction Codes

For reliable transmission and storage of digital data, error detection and correction is required. Below are a few examples of codes which permit error detection and error correction after detection.

Error Detecting Codes

When data is transmitted from one point to another, like in wireless transmission, or it is just stored, like in hard disks and memories, there are chances that data may get corrupted. To detect these data errors, we use special codes, which are error detection codes.

Parity

In parity codes, every data byte, or nibble (according to how user wants to use it) is checked if they have even number of ones or even number of zeros. Based on this information an additional bit is appended to the original data. Thus if we consider 8-bit data, adding the parity bit will make it 9 bit long.

At the receiver side, once again parity is calculated and matched with the received parity (bit 9), and if they match, data is ok, otherwise data is corrupt.

There are two types of parity:

- **Even parity:** Checks if there is an even number of ones; if so, parity bit is zero. When the number of ones is odd then parity bit is set to 1.
- **Odd Parity:** Checks if there is an odd number of ones; if so, parity bit is zero. When number of ones is even then parity bit is set to 1.

Other than the parity various types of check sum techniques are used to detect the error in the transmitted data such as, Adding all bytes, CRC, Fletcher's checksum, Adler-32, etc.,

Error-Correcting Codes

Error-correcting codes not only detect errors, but also correct them. This is used normally in Satellite communication, where turn-around delay is very high as is the probability of data getting corrupt.

Hamming Code

Hamming code adds a minimum number of bits to the data transmitted in a noisy channel, to be able to correct every possible one-bit error. It can detect (not correct) two-bit errors and cannot distinguish between 1-bit and 2-bits inconsistencies. In general it can't detect 3(or more)-bits errors. The hamming detection code is formed based on the parity.

The number of parity bits (p) has to be used in hamming code for the given number of data bits (m) can be found with the help of the following relation,

$$2^p \geq p + m + 1$$

General algorithm

The following general algorithm generates a single-error correcting (SEC) code for any number of bits.

1. Number the bits starting from 1: bit 1, 2, 3, 4, 5, etc.
2. Write the bit numbers in binary. 1, 10, 11, 100, 101, etc.
3. All bit positions that are powers of two (have only one 1 bit in the binary form of their position) are parity bits.
4. All other bit positions, with two or more 1 bits in the binary form of their position, are data bits.
5. Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.
    1. Parity bit 1 covers all bit positions which have the least significant bit set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.

2. Parity bit 2 covers all bit positions which have the second least significant bit set: bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.
3. Parity bit 4 covers all bit positions which have the third least significant bit set: bits 4–7, 12–15, 20–23, etc.
4. Parity bit 8 covers all bit positions which have the fourth least significant bit set: bits 8–15, 24–31, 40–47, etc.
5. In general each parity bit covers all bits where the binary AND of the parity position and the bit position is non-zero.

The form of the parity is irrelevant. Even parity is simpler from the perspective of theoretical mathematics, but there is no difference in practice.

Parity Advantages

The advantages of Parity are,

- Single bit error can be detected.
- It very much easier to form.
- It requires only one bit to detect the error.
- It act as is the base for other types of error detection and correction codes like Hamming code.

Magnitude Comparator

Another common and very useful combinational logic circuit is that of the Magnitude **Comparator** circuit. Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs.

For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of *Boolean Algebra*. There are two main types of Magnitude **Comparator** available and these are.

- Identity Comparator – an *Identity Comparator* is a digital comparator that has only one output terminal for when A = B either "HIGH" A = B = 1 or "LOW" A = B = 0
- Magnitude Comparator – a *Magnitude Comparator* is a digital comparator which has three output terminals, one each for equality, A = B greater than,A > B and less than A < B

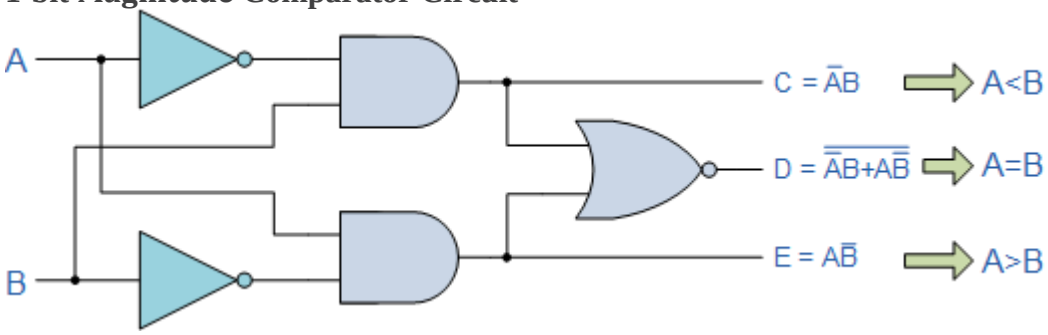
The purpose of a Magnitude **Comparator** is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, …. An, etc) against that of a constant or unknown value such as B (B1, B2, B3, …. Bn, etc) and produce an output condition or flag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other.

$$A > B, \quad A = B, \quad A < B$$

Which means: A is greater than B, A is equal to B, and A is less than B

This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below.

1-bit Magnitude Comparator Circuit



Then the operation of a 1-bit digital comparator is given in the following Truth Table.

Magnitude Comparator Truth Table

Inputs		Outputs		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

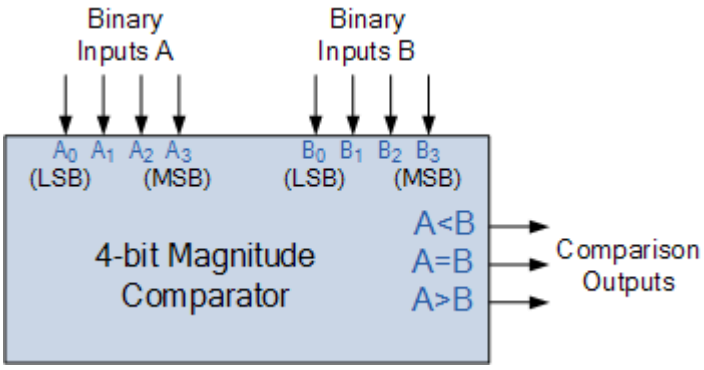
You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two "0" or two "1"'s as an output A = B is produced when they are both equal, either A = B = "0" or A = B = "1". Secondly, the output condition for A = B resembles that of a commonly available logic gate, the Exclusive-NOR or Ex-NOR function (equivalence) on each of the n-bits giving:$Q = A \oplus B$

Digital comparators actually use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary or BCD values or variables against each other, we are comparing the "magnitude" of these values, a logic "0" against a logic "1" which is where the term **Magnitude Comparator** comes from.

As well as comparing individual bits, we can design larger bit comparators by cascading together n of these and produce a n-bit comparator just as we did for the n-bit adder in the previous tutorial. Multi-bit comparators can be constructed to compare whole binary or BCD words to produce an output if one word is larger, equal to or less than the other.

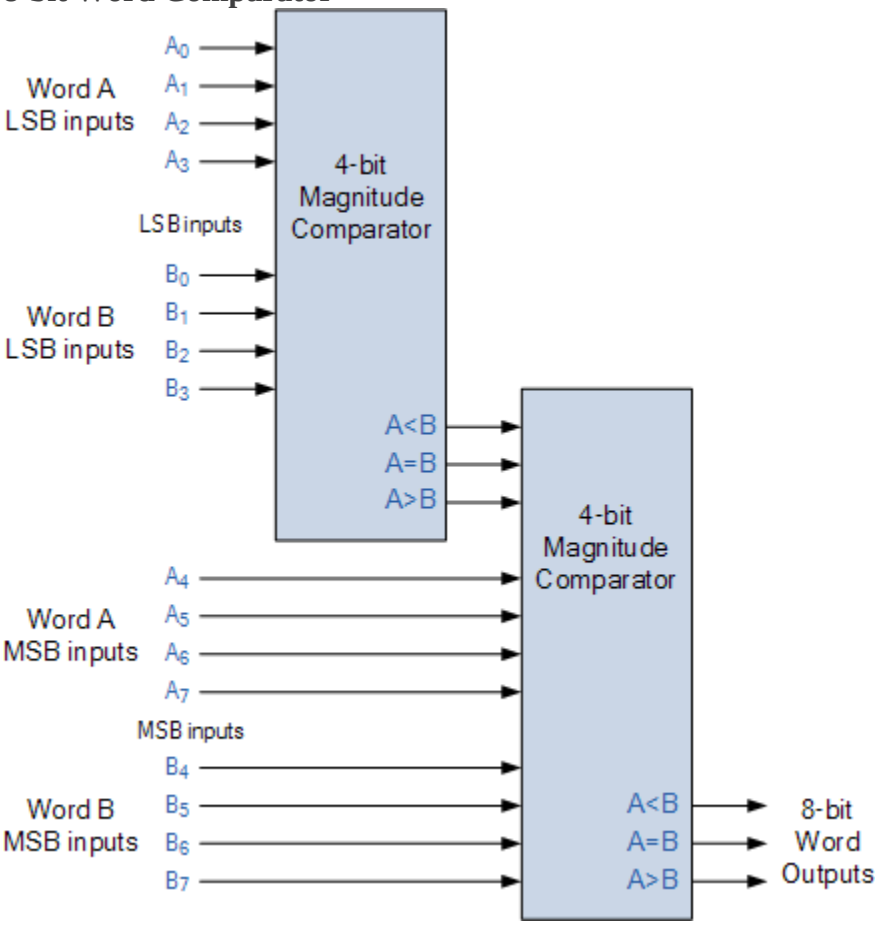
A very good example of this is the 4-bit **Magnitude Comparator**. Here, two 4-bit words ("nibbles") are compared to each other to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown below.

4-bit Magnitude Comparator



Some commercially available digital comparators such as the TTL 74LS85 or CMOS 4063 4-bit magnitude comparator have additional input terminals that allow more individual comparators to be "cascaded" together to compare words larger than 4-bits with magnitude comparators of "n"-bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator as shown to compare 8, 16 or even 32-bit words.

8-bit Word Comparator



When comparing large binary or BCD numbers like the example above, to save time the comparator starts by comparing the highest-order bit (MSB) first. If equality exists, A = B then it

compares the next lowest bit and so on until it reaches the lowest-order bit, (LSB). If equality still exists then the two numbers are defined as being equal.

If inequality is found, either A > B or A < B the relationship between the two numbers is determined and the comparison between any additional lower order bits stops. **Digital Comparator** are used widely in Analogue-to-Digital converters, (ADC) and Arithmetic Logic Units, (ALU) to perform a variety of arithmetic operations.

TEXT BOOKS:

1. Carl Hamacher ,(2012). Computer Organization,(5th ed.). McGraw Hill
2. Dos Reis,A.J.,(2009), Assembly Language and Computer Architecture using C++ and JAVA Technology.
3. M.Mories Mano(2013) Computer Systems Architecture,(3rd ed.), Prentice Hall of India
4. Stalings, W., (2010). Computer Organization and Architecture Designing for Performance,(8th ed.), Prentice Hall of India

PART –A(One Mark Questions- Online Examination)

1. What is the one's complement of $(1011110)_2$
   a. 0100001 b. 0111101 c. 1010000 d. 0001110
2. Convert $(255)_{10}$ to binary number ( ) 2
   a. 111111110 b.100101010 c. 101010101 d. 010101010
3. Signed number is used to indicate ____ numbers
   a. negative b. positive c. negative and positive d. zero
4. The addressing mode, in which operand value specified directly is _____.
   a) Immediate b) Direct c) Definite d) Relative
5. To get the physical address from the logical address generated by CPU we use ____.
   a) MAR b) MMU c) Overlays d) TLB
6. The _____ cycle is a hardware implementation of a branch and save return address operation
   a) Instruction b) timing c) interrupt d) memory
7. The program counter consists of _____ bits
   a) 16 b) 8 c) 11 d) 12
8. When the CPU detects an interrupt, it then saves its
   a. previous state b. next state c. current state d. future state
9. A microprogram is sequencer perform the operation____
   a. read b. write c. read and write d. read and execute
10. A computer program that converts an entire program into machine language at one time is called
    a. interpreter b. simulator c. compiler d. commander
11. A computer program that converts an assembly program into machine language at one time is called
    a. interpreter b. assembler c. compiler d. commander
12. Interrupts which are initiated by an instruction are___
    a. internal b. external c. hardware d. software

PART – B(2 Marks Questions)

1. Write short notes on interrupt.
2. What is an assembly language?
3. What is ROM and RAM?
4. Write short notes on interrupt.
5. What is a machine language?

PART - C (6 Marks Questions)

1. Explain 1's complement and 2's complement number with examples.
2. Convert the following numbers, i. $(1246)_8$=($)_2$ ii. $(AB67)_{16}$ = ($)_2$
3. Explain adder and subtractor with examples.
4. Convert the following numbers, i. $(1246)_8$=($)_2$ ii. $(ABCF7)_{16}$ = ($)_2$
5. Write in detail about various addressing modes.
6. Explain the architecture of a basic Computer.
7. Explain binary addition and subtraction with example.

UNIT-3

Computer Registers

A register is a very small amount of very fast memory that is built into the CPU (central processing unit) in order to speed up its operations by providing quick access to commonly used values. Registers refers to semiconductor devices whose contents can be accessed (i.e., read and written to) at extremely high speeds but which are held there only temporarily (i.e., while in use or only as long as the power supply remains on).

Registers are the top of the memory hierarchy and are the fastest way for the system to manipulate data.Registers are normally measured by the number of bits they can hold, for example, an 8-bit register means it can store 8 bits of data or a 32-bit register means it can store 32 bit of data.

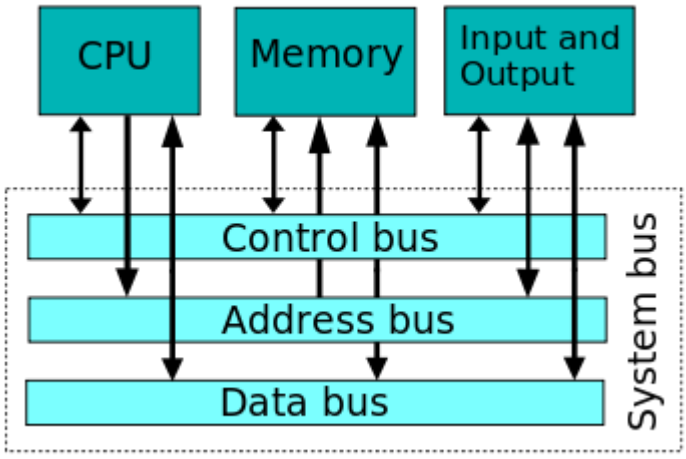
Registers are used to store data temporarily during the execution of a program. Some of the registers are accessible to the user through instructions. Data and instructions must be put into the system. So we need registers for this.

The basic computer registers with their names, size and functions are listed below

Register Symbol	Register Name	Number of Bits	Description
AC	Accumulator	16	Processor Register
DR	Data Register	16	Hold memory data
TR	Temporary Register	16	Holds temporary Data
IR	Instruction Register	16	Holds Instruction Code
AR	Address Register	12	Holds memory address
PC	Program Counter	12	Holds address of next instruction
INPR	Input Register	8	Holds Input data
OUTR	Output Register	8	Holds Output data

System Bus

A **system bus** is a single computer bus that connects the major components of a computer system, combining the functions of adata bus to carry information, an address bus to determine where it should be sent, and a control bus to determine its operation. The technique was developed to reduce costs and improve modularity, and although popular in the 1970s and 1980s, more modern computers use a variety of separate buses adapted to more specific needs.



Computer Instructions

Computer instructions are the basic components of a machine language program. They are also known as *macrooperations*, since each one is comprised of a sequences of microoperations.

Each instruction initiates a sequence of microoperations that fetch operands from registers or memory, possibly perform arithmetic, logic, or shift operations, and store results in registers or memory.

Instructions are encoded as binary *instruction codes*. Each instruction code contains of a *operation code*, or *opcode*, which designates the overall purpose of the instruction (e.g. add, subtract, move, input, etc.). The number of bits allocated for the opcode determined how many different instructions the architecture supports.

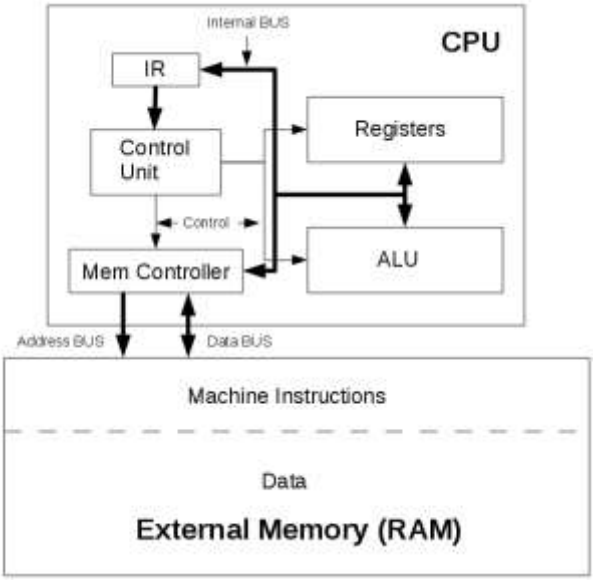
In addition to the opcode, many instructions also contain one or more *operands*, which indicate where in registers or memory the data required for the operation is located. For example, and add instruction requires two operands, and a not instruction requires one.

```
 15    12 11         6 5         0
+-----------------------------------+
| Opcode |  Operand    |  Operand   |
+-----------------------------------+
```

The opcode and operands are most often encoded as unsigned binary numbers in order to minimize the number of bits used to store them. For example, a 4-bit opcode encoded as a binary number could represent up to 16 different operations.

The *control unit* is responsible for decoding the opcode and operand bits in the instruction register, and then generating the control signals necessary to drive all other hardware in the CPU to perform the sequence of microoperations that comprise the instruction.

CPU Block Diagram



. Effective Address by Addressing Mode

Mode	**Effective Address**
Immediate	Address of the instruction itself
Direct	Address contained in the instruction code
Indirect	Address at the address in the instruction code

Basic Computer Instruction Format

The Basic Computer has a 16-bit instruction code similar to the examples described above. It supports direct and indirect addressing modes.

How many bits are required to specify the addressing mode?

```
 15 14 12 11      0
+------------------+
```

```
| I | OP | ADDRESS |
+-----------------+
```

I = 0: direct
I = 1: indirect

Computer Instructions

All Basic Computer instruction codes are 16 bits wide. There are 3 instruction code formats:

- Memory-reference instructions take a single memory address as an operand, and have the format:

```
 15  14 12 11      0
+-------------------+
| I | OP  | Address |
+-------------------+
```

If I = 0, the instruction uses direct addressing. If I = 1, addressing in indirect.

How many memory-reference instructions can exist?

- Register-reference instructions operate solely on the AC register, and have the following format:

```
 15  14 12 11     0
+-----------------+
| 0 | 111 | OP    |
+-----------------+
```

How many register-reference instructions can exist? How many memory-reference instructions can coexist with register-reference instructions?

- Input/output instructions have the following format:

```
 15  14 12 11     0
+-----------------+
| 1 | 111 | OP    |
+-----------------+
```

Timing and Control

All sequential circuits in the Basic Computer CPU are driven by a master clock, with the exception of the INPR register.

At each clock pulse, the control unit sends control signals to control inputs of the bus, the registers, and the ALU.

Control unit design and implementation can be done by two general methods:

- A *hardwired* control unit is designed from scratch using traditional digital logic design techniques to produce a minimal, optimized circuit. In other words, the control unit is like an ASIC (application-specific integrated circuit).
- A *microprogrammed* control unit is built from some sort of ROM. The desired control signals are simply stored in the ROM, and retrieved in sequence to drive the microoperations needed by a particular instruction.

Instruction Cycle

In this chapter, we examine the sequences of microoperations that the Basic Computer goes through for each instruction. Here, you should begin to understand how the required control signals for each state of the CPU are determined, and how they are generated by the control unit.

The CPU performs a sequence of microoperations for each instruction. The sequence for each instruction of the Basic Computer can be refined into 4 abstract phases:

1. Fetch instruction
2. Decode
3. Fetch operand
4. Execute

Program execution can be represented as a top-down design:

1. Program execution
    a. Instruction 1
        i. Fetch instruction
        ii. Decode
        iii. Fetch operand
        iv. Execute
    b. Instruction 2
        i. Fetch instruction
        ii. Decode
        iii. Fetch operand
        iv. Execute
    c. Instruction 3 ...

Program execution begins with:

PC ← address of first instruction, SC ← 0

After this, the SC is incremented at each clock cycle until an instruction is completed, and then it is cleared to begin the next instruction. This process repeats until a HLT instruction is executed, or until the power is shut off.

Instruction Fetch and Decode

The instruction fetch and decode phases are the same for all instructions, so the control functions and microoperations will be independent of the instruction code.

Everything that happens in this phase is driven entirely by timing variables T_0, T_1 and T_2. Hence, all control inputs in the CPU during fetch and decode are functions of these three variables alone.

T_0: AR ← PC

T_1: IR ← M[AR], PC ← PC + 1

T_2: $D_{0\text{-}7}$ ← decoded IR(12-14), AR ← IR(0-11), I ← IR(15)

For every timing cycle, we assume SC ← SC + 1 unless it is stated that SC ← 0.

The operation $D_{0\text{-}7}$ ← decoded IR(12-14) is not a register transfer like most of our microoperations, but is actually an inevitable consequence of loading a value into the IR register. Since the IR outputs 12-14 are directly connected to a decoder, the outputs of that decoder will change as soon as the new values of IR(12-14) propagate through the decoder.

Note that incrementing the PC at time T_1 assumes that the next instruction is at the next address. This may not be the case if the current instruction is a branch instruction. However, performing the increment here will save time if the next instruction immediately follows, and will do no harm if it doesn't. The incremented PC value is simply overwritten by branch instructions.

In hardware development, unlike serial software development, it is often advantageous to perform work that may not be necessary. Since we can perform multiple microoperations at the same time, we might was well do everything that *might* be useful at the earliest possible time.

Likewise, loading AR with the address field from IR at T_2 is only useful if the instruction is a memory-reference instruction. We won't know this until T_3, but there is no reason to wait since there is no harm in loading AR immediately.

Memory-Reference

AC ← AC + M[AR]

For the memory-reference execute phase, all control inputs in the CPU are functions of timing signals T_4 or later, I, and one of the variables D_0 through D_6.

The execute phase for memory-reference instructions begins at time T_4. The effective address was loaded into AR at time T_2 or T_3.

Several memory-reference instructions operate on AC and an operand from memory. Since we cannot feed data directly from memory into the ALU, we need to refine the register transfer statements from table 5-4 into microoperations.

AND

The AND instruction is indicated by signal D_0.

D_0T_4: DR ← M[AR]

D_0T_5: AC ← AC ^ DR, SC ← 0

Show the Boolean functions and circuits for all control inputs required to carry out these microoperations.

BSA

Symbolic notation:

M[AR] ← PC, PC ← AR + 1

Control functions and microoperations:

D_5T_4: M[AR] ← PC, AR ← AR + 1

D_5T_5: PC ← AR, SC ← 0

ISZ

ISZ is useful for implementing loops:

```
LOOP_COUNT, DEC -10
X,        DEC 0

          LDA    LOOP_COUNT  / Initialize loop counter
          STA    X
LOOP,

          ISZ    X          / Check loop condition
          BUN    LOOP
```

D_6T_4: DR ← M[AR]

D_6T_5: DR ← DR + 1

D_6T_6: M[AR] ← DR, SC ← 0

D_6T_6DR(0-15)': PC ← PC + 1

Note

DR(0-15) = (DR(0) ^ DR(1) ^ DR(2) ^ ... ^ DR(15))

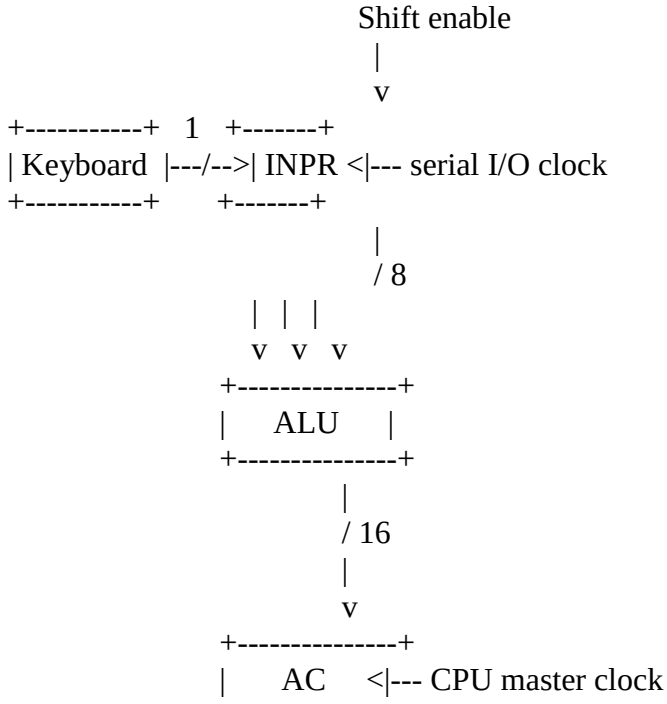
Show the Boolean functions and circuits necessary to implement this instruction.

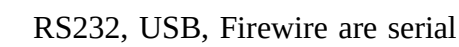
Input-Output and Interrupt

Hardware Summary

The Basic Computer I/O consists of a simple terminal with a keyboard and a printer/monitor.

The keyboard is connected serially (1 data wire) to the INPR register. INPR is a shift register capable of shifting in external data from the keyboard one bit at a time. INPR outputs are connected in parallel to the ALU.





RS232, USB, Firewire are serial interfaces with their own clock independent of the CPU. (USB speed is independent of processor speed.)

- RS232: 115,200 kbps (some faster)
- USB: 11 mbps
- USB2: 480 mbps
- FW400: 400 mbps
- FW800: 800 mbps
- USB3: 4.8 gbps

OUTR inputs are connected to the bus in parallel, and the output is connected serially to the terminal. OUTR is another shift register, and the printer/monitor receives an end-bit during each clock pulse.

I/O Operations

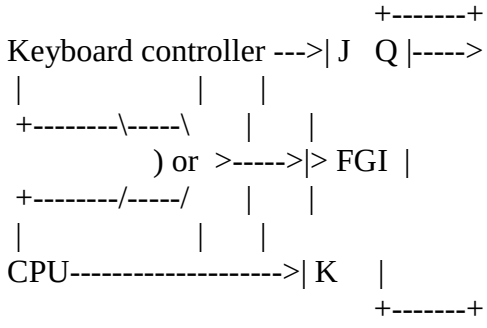
The input and output devices are not under the full control of the CPU (I/O events are asynchronous), the CPU must somehow be told when an input device has new input ready to send, and an output device is ready to receive more output.

The FGI flip-flop is set to 1 after a new character is shifted into INPR. This is done by the I/O interface, not by the control unit. This is an example of an asynchronous input event (not synchronized with or controlled by the CPU).

The FGI flip-flop must be cleared after transferring the INPR to AC. This must be done as a microoperation controlled by the CU, so we must include it in the CU design.

The FGO flip-flop is set to 1 by the I/O interface after the terminal has finished displaying the last character sent. It must be cleared by the CPU after transferring a character into OUTR.

Since the keyboard controller only sets FGI and the CPU only clears it, a JK flip-flop is convenient:



There are two common methods for detecting when I/O devices are ready, namely *software polling* and *interrupts*. These two methods are discussed in the following sections.

Software Polling

In software polling, the software is responsible for checking the status of I/O devices and initiating transactions when the device is ready. The simplest form of software polling is called *spin waiting*. A *spin waiting* loop does nothing but watch the status of a device until it becomes ready. When it is ready, the loop exits and the I/O transaction is performed.

```
key_wait,   ski
            bun     key_wait
            inp
            / Do something with the input
```

Interrupts

Introduction About program interrupt:-

When a Process is executed by the CPU and when a user Request for another Process then this will create disturbance for the Running Process. This is also called as the **Interrupt**.

Interrupts can be generated by User, Some Error Conditions and also by Software's and the hardware's. But CPU will handle all the Interrupts very carefully because when Interrupts are generated then the CPU must handle all the Interrupts Very carefully means the CPU will also Provides Response to the Various Interrupts those are generated. So that When an interrupt has Occurred then the CPU will handle by using the Fetch, decode and Execute Operations.

Interrupts allow the operating system to take notice of an external event, such as a mouse click. Software interrupts, better known as exceptions, allow the OS to handle unusual events like divide-by-zero errors coming from code execution.

The sequence of events is usually like this:

1. Hardware signals an interrupt to the processor
2. The processor notices the interrupt and suspends the currently running software
3. The processor jumps to the matching interrupt handler function in the OS
4. The interrupt handler runs its course and returns from the interrupt
5. The processor resumes where it left off in the previously running software

The most important interrupt for the operating system is the timer tick interrupt. The timer tic interrupt allows the OS to periodically regain control from the currently running user process. The OS can then decide to schedule another process, return back to the same process, do housekeeping, etc. The timer tick interrupt provides the foundation for the concept of preemptive multitasking.

Types of Interrupts

Generally there are three types of Interrupts those are Occurred For Example

1) Internal Interrupt

2) External Interrupt.

3) Software Interrupt.

1.Internal Interrupt:-

• When the hardware detects that the
program is doing something wrong, it will
usually generate an interrupt usually generate an interrupt.
– Arithmetic error - Invalid Instruction
– Addressing error - Hardware malfunction
– Page fault - Debugging
• A Page Fault interrupt is not the result of a program
error, but it does require the operating system to get
control.
• Internal interrupts are sometimes called
exceptions

The Internal Interrupts are those which are occurred due to Some Problem in the Execution For Example When a user performing any Operation which contains any Error and which contains any type of Error. So that Internal Interrupts are those which are occurred by the Some Operations or by Some Instructions and the Operations those are not Possible but a user is trying for that Operation. And The Software Interrupts are those which are made some call to the System for Example while we are Processing Some Instructions and when we wants to Execute one more Application Programs.

2.External Interrupt:-

• I/O devices tell the CPU that an I/O
request has completed by sending an
interrupt signal to the processor.
• I/O errors may also generate an interrupt.
• Most computers have a timer which
interrupts the CPU every so many interrupts the CPU every so many
milliseconds.

The External Interrupt occurs when any Input and Output Device request for any Operation and the CPU will Execute that instructions first For Example When a Program is executed and when we move the Mouse on the Screen then the CPU will handle this External interrupt first and after that he will resume with his Operation.

3.Software interrupts:-

These types if interrupts can occur only during the execution of an instruction. They can be used by a programmer to cause interrupts if need be. The primary purpose of such interrupts is to switch from user mode to supervisor mode.

A software interrupt occurs when the processor executes an INT instruction. Written in the program, typically used to invoke a system service.

A processor interrupt is caused by an electrical signal on a processor pin. Typically used by devices to tell a driver that they require attention. The clock tick interrupt is very common, it wakes up the processor from a halt state and allows the scheduler to pick other work to perform.

A processor fault like access violation is triggered by the processor itself when it encounters a condition that prevents it from executing code. Typically when it tries to read or write from unmapped memory or encounters an invalid instruction.

With interrupts, the running program is not responsible for checking the status of I/O devices.

When a device becomes ready, the CPU *hardware* initiates a branch to an I/O subprogram called an *interrupt service routine (ISR),* which handles the I/O transaction with the device.

An interrupt can occur during *any* instruction cycle as long as interrupts are enabled. When the current instruction completes, the CPU interrupts the flow of the program, executes the ISR, and then resumes the program. The program itself is not involved and is in fact unaware that it has been interrupted.

Interrupts can be globally enabled or disabled via the IEN flag (flip-flop).

Some architectures have a separate ISR for each device. The Basic Computer has a single ISR that services both the input and output devices.

If interrupts are enabled, then when either FGI or FGO gets set, the R flag also gets set. (R = FGI v FGO) This allows the system to easily check whether *any* I/O device needs service. Determining which one needs service can be done by the ISR.

If R = 0, the CPU goes through a normal instruction cycle. If R = 1, the CPU branches to the ISR to process an I/O transaction.

nterrupts are usually disabled while the ISR is running, since it is difficult to make an ISR *reentrant*. (Callable while it is already in progress, such as a recursive function.) Hence, IEN and R are cleared as part of the interrupt cycle. IEN should be re-enabled by the ISR when it is finished.

Design of Basic Computer

To develop the entire control unit, we must determine the Boolean function for every control input on the bus, every register, the ALU, and the various flip-flops such as the I bit, the IEN flag, etc.

The independent variables for each of these functions include the timing signals T_0 through T_{16}, the decoded opcode, the I bit, and so on.

Interconnection Structures

Introduction

Different types of exchanges are required for communication in a computer. Figure 5.1 shows the Memory Module, I/O Module and CPU Module and also indicates the major forms of inputs and outputs of these modules.

Types of exchange of information

A computer consists of three basic types of modules (processor, memory, I/O) that communicate with each other. Thus, there must be paths for connecting the modules. The collection of paths is called the **"interconnection structure"**. The design of this structure will depend on the exchanges that must be made between modules.

Modules of a system

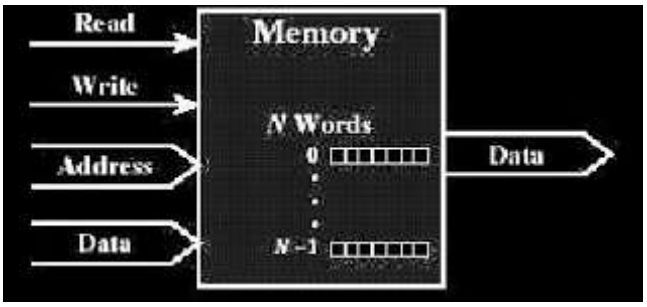


Figure 5.1: a) Memory module

Memory module consists of N words of equal length. Each word is assigned a unique numerical value (0, 1, . . ., N-1). Figure 5.1(a) shows the possible inputs and output of a memory module. A word of data can be read from or written into the memory depending on whether the control signal is Memory Read (MR) or Memory Write (MW). The location of the memory is provided by the input called as Address.

Example: List the inputs and outputs that are necessary to write the data to the memory.

Solution:

The inputs that are required are

1. Control signal which is memory write,

2. Address that gives the location of the memory where the data is to be written and the data itself.

There is no output required for this particular task.

The sequence of operations that take place:

1. The position of the memory is located using the input address.

2. Then it sees the input control signal is MW and then the input which is data is placed in the location of the memory.

I/O Module

I/O is functionally similar to memory. The possible inputs and outputs for a typical I/O module is as shown in figure 5.1(b).

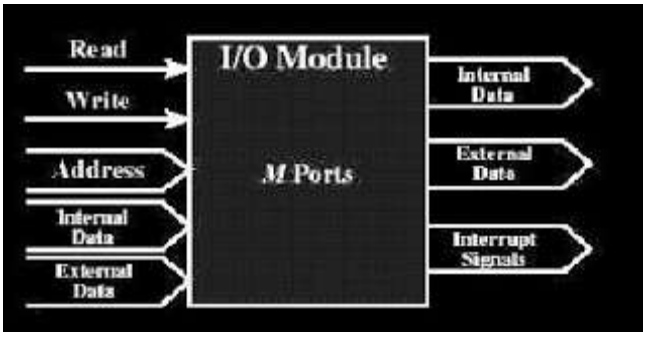


Figure 5.1: b) I/O Module

There are two operations **read** and **write**. I/O module may control more than one external device. We usually refer to each of the interface of the external device as a ***port***. Each port is

given a unique address as 0, 1, ……, M-1. Also there are external data paths for the input and output of data with an external device. Also an I/O module may be able to send interrupt signals to the CPU.

CPU Module

The possible inputs and outputs for a typical CPU module is as shown in figure 5.1(c). The CPU reads in the instructions and data and writes out the data after processing. It uses control signals to control the overall operation of the system. It also receives the interrupt signals and appropriate actions to be taken in outputs data as well as control signals.

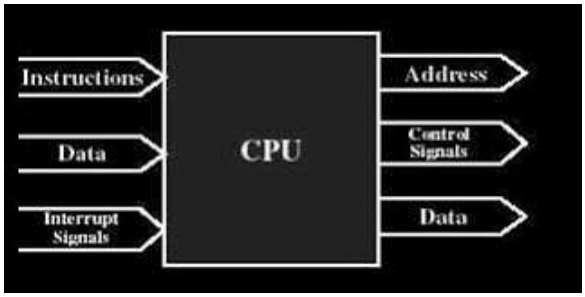


Figure 5.1: c) CPU Module

Different types of transfers

There are two types of transfers that are classified; one depending on the devices that are to be interconnected and the other depending on whether it carries information one bit or several bits.

Types of transfers between different modules

Following are the possible ways of transfer of information between the three modules of the system:

· **Memory to Processor (CPU):** The processor reads an instruction or a unit of data from memory.

· **CPU to Memory:** The processor writes a unit of data to memory.

· **I/O to CPU:** The processor reads data from an I/O device via an I/O module.

· **CPU to I/O:** The processor sends data to the I/O device.

· **I/O to or from memory:** For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using **Direct Memory Access (DMA).**

Serial & Parallel transfer

A bus consists of multiple communication lines (or pathways). Each line is capable of transmitting signals representing binary 1 or binary 0. Bits are transmitted in Serial & Parallel. When only one bit is carried at a time it requires only a single wire as shown in figure 5.2. When we consider many bits to be transmitted at a time it requires many wires as shown in figure 5.3. These many wires together constitute a bus and the mode of transmission is termed as parallel transmission.

Figure 5.2: Serial transmission - one bit at a time

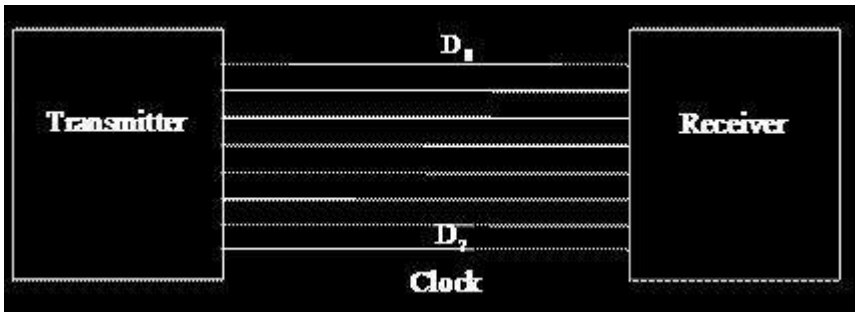


Figure 5.3: Parallel transmission - several (8 bits here) bits at a time

Table gives the Comparison of serial & parallel transmission

Factor	*Serial mode*	*Parallel mode*
Cost	Less costly (only one wire)	More costly (many wires)
Speed	Low (only 1 bit at a time)	High (more bits at a time)
Throughput	Low	High

5.3 Types of Buses

As discussed earlier in unit 1, a system bus consists of a ***data bus,*** **a** ***memory address bus*** **and a** ***control bus***. The interconnection between the modules of a system is as shown in figure 5.4

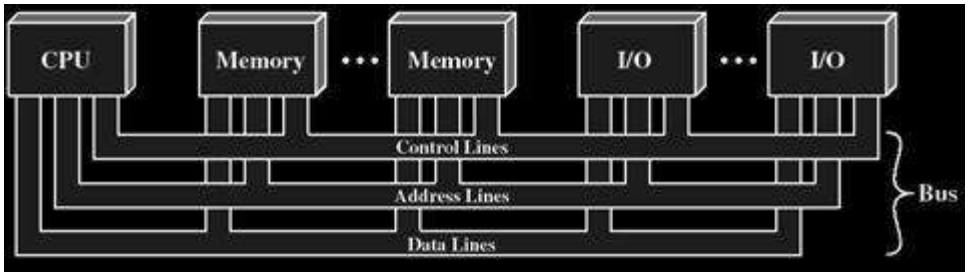


Figure 5.4: Bus Interconnection scheme

Data Bus: A bus, which carries a word to or from memory, is called ***data bus***. Its width is equal to the word length of the memory. Also, it provides a means for moving data between the different modules of a system. The data bus usually consists of 8, 16 or 32 separate lines. The number of lines implies the data bus

Address bus: A bus that is used to carry the address of the data in the memory and its width is equal to the number of bits in the **Memory Address Register (MAR)** of the memory.

Example: If a computer memory has 64K, 32-bit words, then the data bus will be 32-bits wide and the address bus will be 16-bits wide.

Control bus: A bus that is used to control the access carries the control signals between the various units of the computer. The processor has to send commands READ and WRITE to the memory which requires single wire. A START command is necessary for the I/O units. All these signals are carried by the control bus.

Types of Control Lines

· **Memory Write:** Causes data on the bus (data bus) to be written into the addressed location.

· **Memory Read:** Causes data from the addressed location to be placed on the bus (data bus).

· **I/O Write:** Causes data on the data bus to be output to the addressed I/O port.

· **I/O Read:** Causes data from the addressed I/O port to be placed on the bus (data bus).

· **Transfer ACK:** Indicates that data has been accepted from or placed on the bus.

· **Bus Request:** Indicates that a module needs to gain control of the bus.

· **Bus Grant:** Indicates that a requesting module has been granted control of the bus.

· **Interrupt Request:** Indicates that an interrupt is pending.

· **Interrupt ACK:** Acknowledges that the pending interrupt has been recognized.

· **Clock:** Used to synchronize operations.

· **Reset:** Initializes all modules.

5.4 Elements of Bus Design

Bus Types: Bus lines can be separated into two types.

· **Dedicated:** Permanently assigned to either one function or to a physical subset of components.

-Functional dedication: Bus has a specific function.

Example: Three busses identified for carrying address, data, and control signals as seen earlier. They are **Address Bus, Data Bus**, and **Control Bus**.

-Physical dedication: Refers to the use of multiple buses, each of which connects only a subset of components using the bus.

Example: I/O buses are used only to interconnect all I/O modules. And this bus is then connected to the main bus through some type of an I/O adapter module.

Advantage of Physical dedication:

It offers high throughput because there is less bus contention.

Disadvantage of Physical dedication:

Increased size and cost of the system

· **Multiplexed:** These are also referred to as **non-dedicated**. Same bus may be used for various functions. The method of using the same bus for multiple purposes is known as **Time Multiplexing**.

Example: Discuss the steps of actions that are to be performed so that address and data information may be transmitted over the same set of lines.

Solution: Assume an additional control signal called address line activation line or **Address Line Enable (ALE)** line is used.

List of operations are as follows:

1. At the beginning of the data transfer the address is placed on the bus with the ALE line activated.

2. Each module is given sufficient period of time to copy the address and determine if it is the addressed module.

3. The address is then removed from the bus, and then same bus connections are used for subsequent read and write data transfer with the ALE signal deactivated.

Advantages: Multiplexing uses fewer lines, which saves space and cost.

Disadvantages of multiplexing:

1. More complex circuitry required in each module.

2. There is potential reduction in the performance as certain events that share the same bus cannot take place in parallel.

Bus Timing

· **Synchronous:** The occurrence of events on the bus is determined by a clock. The bus includes a clock line. A single 1-0 transmission on clock signal is referred to as 1 clock cycle" or "bus cycle", and defines a time slot. Most events occupy a single clock cycle, but some requires more cycles.

· **Asynchronous:** The occurrence of one event on a bus follows and depends on the occurrence of a previous event.

Synchronous timing is simpler to implement and test; however it is less flexible.

With asynchronous timing, a mixture of slow and fast devices can share a bus.

Bus Width:

· **Bus Width of Address Lines:** Number of memory units that can be addressed.

· **Bus Width of Data Lines:** Size of memory units that can be addressed. (8, 12, 16, 32, 64 bits)

Ask and give examples: How many memory addresses with 24 bits, 32 bits, etc.

Bus Speed:

One of the important attribute of busses is its speed. The speed of the bus refers to how fast you can change the data on the bus, and still have devices to be able to read the values correctly. Bus speed can limit how fast a CPU can communicate with memory. The size of a bus can also limit the speed too.

Example: The speed can be measured in say, MHz that is up to 10^6 changes per second.

Data Transfer Type:

· **Read:** (Slave to Master)

· **Write:** (Master to Slave)

· **Read-Modify-Write:** A read followed immediately by a write to the same address. Usually indivisible operation to prevent any access to data by other potential bus masters.

· **Read-After-Write:** Indivisible operation consisting of a write followed immediately by a read of the same address. Generally for checking purposes.

· **Block:** In this case, one address cycle is followed by n data cycles. The first data item is transferred to/from the specified address, the remaining data items are transferred to/from subsequent addresses.

5.5 Bus Structure

If a large number of devices are connected to the bus, the performance will suffer. There are two main causes:

1. In general, the more devices attached to the bus, the greater is the bus length, and hence the greater is the propagation delay.

2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.

To overcome these problems, in most of the modern computers there are multiple buses.

Single Bus System

In this type of inter-connection, the three units share a single bus. Hence the information can be transferred only between two units at a time. Here the I/O units use the same memory address space. This simplifies programming of I/O units as no special I/O instructions are needed. This is one of advantages of single bus organization.

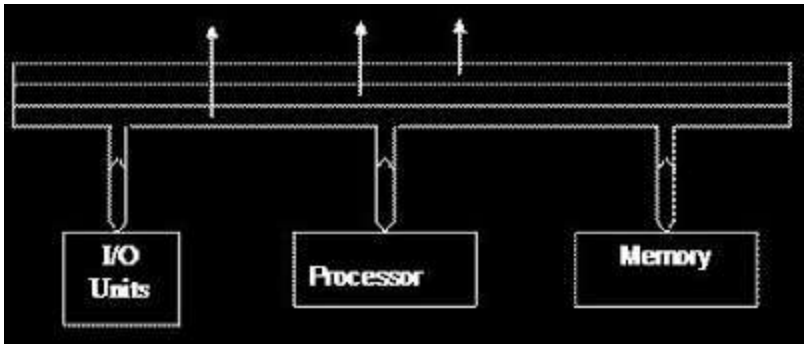


Figure 5.5: Single-Bus Organization

The transfer of information over a bus cannot be done at a speed comparable to the operating speed of all the devices connected to the bus. Some electromechanical devices such as keyboards and printers are very slow whereas disks and tapes are considerably faster. Main memory and processors operate at electronic speeds. Since all the devices must communicate over the bus, it is necessary to smooth out the differences in timings among all the devices.

A common approach is to include ***buffer register*** with the devices to hold the information during transfers. To illustrate this let us take one example. Consider the transfer of an encoded character from the processor to a character printer where it is to be printed. The processor sends the character to the printer output register over the bus. Since buffer is an electronic register this transfer requires relatively little time. Now the printer starts printing. At this time bus and the processor are no longer needed and can be released for other activities. Buffer register is not available for other transfers until the process is completed. Thus buffer register smoothes out the timing differences between the processor, memory and I/O devices. This allows the processor to switch rapidly from one device to another interweaving its processing activity with data transfer involving several I/O devices.

Two Bus Organization

Figure 5.6 shows inter-connection of various computer units through two independent system buses. Here the I/O units are connected to the processor through an I/O bus and the processor is connected to the memory through the memory bus.

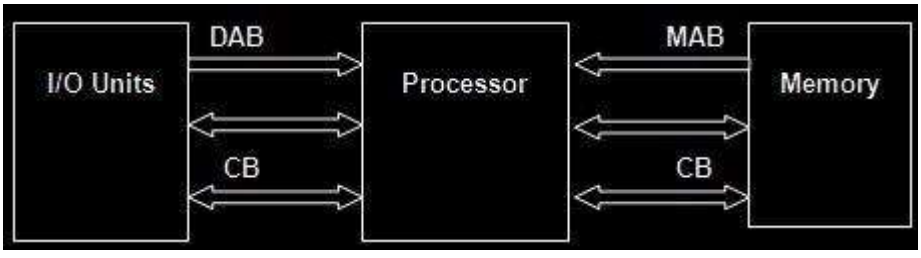


Figure 5.6: Two Bus Organization

The I/O bus consists of device address bus, data bus and a control bus. Device address bus carries the address of the I/O units to be accessed by the processor. The data bus carries a word from the addressed input unit to the processor and from the processor to the addressed output unit. The control bus carries control commands such as START, STOP etc., from the processor to I/O units and it also carries status information of I/O units to the processor. Memory bus also consists of a Memory Address Bus (MAB), data bus and a control bus.

In this type of inter-connection the processor completely supervises the transfer of information to and from the I/O units. All the information is first taken to the processor and from there to the memory. Such a data transfer is called as program controlled transfer.

An alternative two bus structure:

There is one more method to connect processor to the I/O units. Figure 5.7 shows an alternative two bus structure. Here the I/O units are directly connected to the memory.

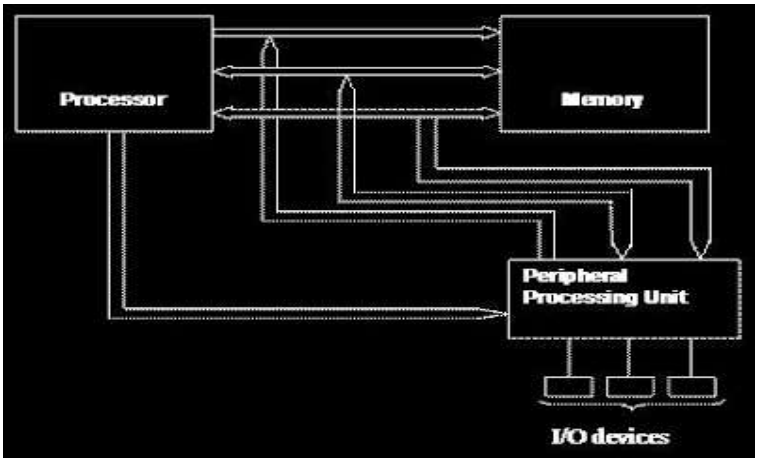


Figure 5.7: Alternative Two-Bus Organization

Here the I/O devices are connected to special interface logic known as ***Direct Memory Access*** **(DMA)** logic or an ***I/O channel***. This is also called as ***Peripheral Processor Unit (PPU).*** The processor issues a READ or WRITE command giving the device address, the address of the memory location where the data read from the input unit is to be stored or from where the data is to be taken to output units, and the number of data words to be transferred. This command is accepted by the PPU, which now takes the responsibility of data transfer.

TEXT BOOKS:

1. Carl Hamacher ,(2012). Computer Organization,(5th ed.). McGraw Hill
2. Dos Reis,A.J.,(2009), Assembly Language and Computer Architecture using C++ and JAVA Technology.
3. M.Mories Mano(2013) Computer Systems Architecture,(3rd ed.), Prentice Hall of India
4. Stalings, W., (2010). Computer Organization and Architecture Designing for Performance,(8th ed.), Prentice Hall of India

PART –A(One Mark Questions- Online Examination)

1. The _____ register is a general purpose processing register.
   a) data b) address c) accumulator d) instruction
2. The basic computer consists of _____ registers.
   a) 5 b) 6 c) 8 d) 9
3. A group of wires used to interconnect the major components is known as
   a) registers b) interrupts c) bus d) instructions
4. _____ method is used to map logical addresses of variable length onto physical memory.
   a) Paging b) Overlays c) Segmentation d) Paging with segmentation
5. A computer program that converts an assembly program into machine language at one time is called
   a. interpreter b. assembler c. compiler d. commander
6. Interrupts which are initiated by an instruction are___
   a. internal b. external c. hardware d. software
7. A simple way of performing I/O tasks is to use a method known as___
   a. program –controlled I/O b. program-controlled input
   c. program-controlled output c. I/O operation
8. An exception conditions in a computer system by an event external to the CPU is called__
   a. interrupt b. halt c. wait d. process
9. Which of the following belongs to an arithmetic instruction?
   a) BUN b) AND c) SKI d) ADD
10. A program residing in the memory unit of the computer consists of a sequence of
    a) Codes b) data c) registers d) instructions
11. Which of the following belongs to a logical instruction?
    HLT b) XOR c) BSA d) DIV

PART – B(2 Marks Questions)

1. What is cache memory?
2. What are the differences between the main memory and control memory?
3. Define cache.
4. Why does the DMA priority over CPU when both request memory transfer
5. What is Flip flop?

PART - C (6 Marks Questions)

1. Describe in detail about the instruction cycle.
2. Discuss in detail about the various instruction set.
3. Explain the functions of input-output and interrupt with neat diagram
4. Discuss in detail about the interconnection structures.
5. Explain the functions of instruction cycle with neat diagram
6. Describe in detail about pipeline processing.
7. Explain in detail about hazards
8. Describe in detail about the types of addressing modes
9. Discuss in detail about the stack organization.

UNIT-4

Register Organization:—

The number of registers in a processor unit may vary from just one processor register to as many as 64 registers or more.

1. One of the CPU registers is called as an accumulator AC or 'A' register. It is the main operand register of the ALU.
2. The data register (DR) acts as a buffer between the CPU and main memory. It is used as an input operand register with the accumulator.
3. The instruction register (IR) holds the opcode of the current instruction.
4. The address register (AR) holds the address of the memory in which the operand resides.
5. The program counter (PC) holds the address of the next instruction to be fetched for execution.

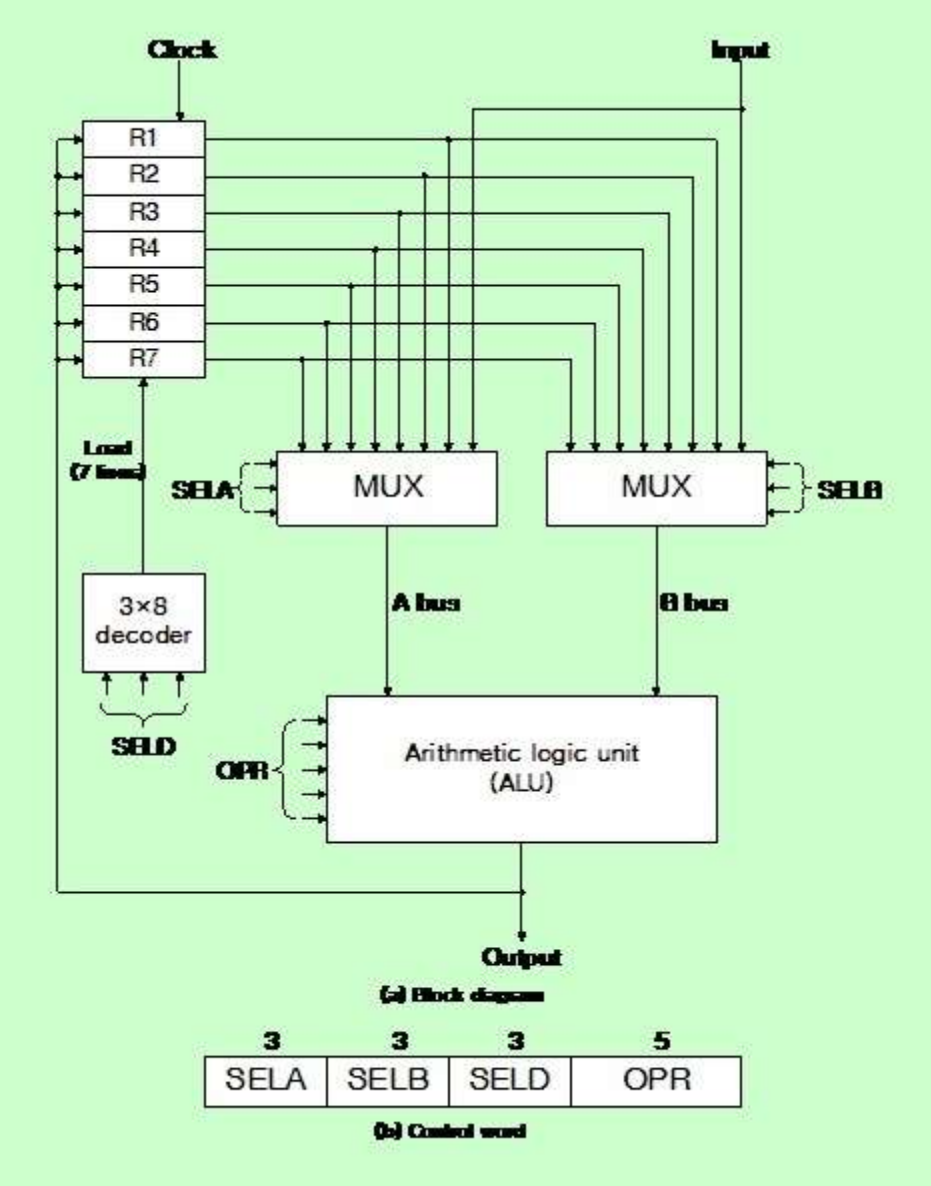
Additional addressable registers can be provided for storing operands and address. This can be viewed as replacing the single accumulator by a set of registers. If the registers are used for many purpose, the resulting computer is said to have general register organization. In the case of processor registers, a registers is selected by the multiplexers that form the buses.

When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfers, but also while performing various micro-operations. Hence it is necessary to provide a common unit that can perform all the arithmetic, logic and shift micro-operation in the processor.

A Bus organization for seven CPU registers:—

The output of each register is connected to true multiplexer (mux) to form the two buses A & B. The selection lines in each multiplexer select one register or the input data for the particular bus. The A and B buses forms the input to a common ALU. The operation selected in the ALU determines the arithmetic or logic micro-operation that is to be performed. The result of the micro-operation is available for output and also goes into the inputs of the registers. The register that receives the information from the output bus is selected by a decoder. The decoder activates one of the register load inputs, thus providing a transfer both between the data in the output bus and the inputs of the selected destination register.

The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the systems.



R1 ® R2 + R3

(1) MUX A selection (SEC A): to place the content of R2 into bus A
(2) MUX B selection (sec B): to place the content of R3 into bus B
(3) ALU operation selection (OPR): to provide the arithmetic addition (A + B)
(4) Decoder destination selection (SEC D): to transfer the content of the output bus into R1

These form the control selection variables are generated in the control unit and must be available at the beginning of a clock cycle. The data from the two source registers propagate through the gates in the multiplexer and the ALU, to the output bus, and the destination registers, all during the clock cycle intervals.

Control Word:-

There are 14 binary selection inputs in the units, and their combined value specified a control word. It consists of four fields three fields contain three bits each, and one field has five bits. The three bits of SEL A select a source register for the A input of the ALU. The three bits of SEL B select a source register for the B input of the ALU. The three bit of SEC D select a destination register using the decoder and its seven load outputs. The five bits of OPR select one of the operations in the ALU. The 14-bit control word when applied to the selection inputs specify a particular micro-operation.

Table: Encoding of Register selection fields.

Binary Code	SEL A	SEL B	SEL D
000	Input	Input	None
001	R1		
010	R2		
011	R3	S	S
100	R4	A	A
101	R5	M	M
110	R6	E	E
111	R7		

Table: Encoding of ALU operation

OPR & select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A-B	SUB
00110	Decrement A	DEC A
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA

11000	Shift left A	SHLA

Examples of Micro-operation for the CPU

Symbolic Designation

Micro Operation	SECA	SEC B	SEL D	OPR	Control Word			
R1 ® R2 – R3	R2	R3	R1	SUB	010	011	001	00101
R4 ® R5 V R5	R4	R	R4	OR	100	101	100	0101
R6 ® R6 + 1	R6	-	R6	MCA	110	000	110	00001
R7 ® R1	R1	-	R7	TSFA	001	000	111	00000
Output ® R2	R2	-	None	TSFA	010	000	000	00000
Output ® Input	Input	-	None	TSFA	000	000	000	00000
R4 ® SHL R4	R4	-	R4	SHLA	100	000	100	11000
R5 ® 0	R5	R5	R5	XOR	101	101	101	01100

Stack Organization

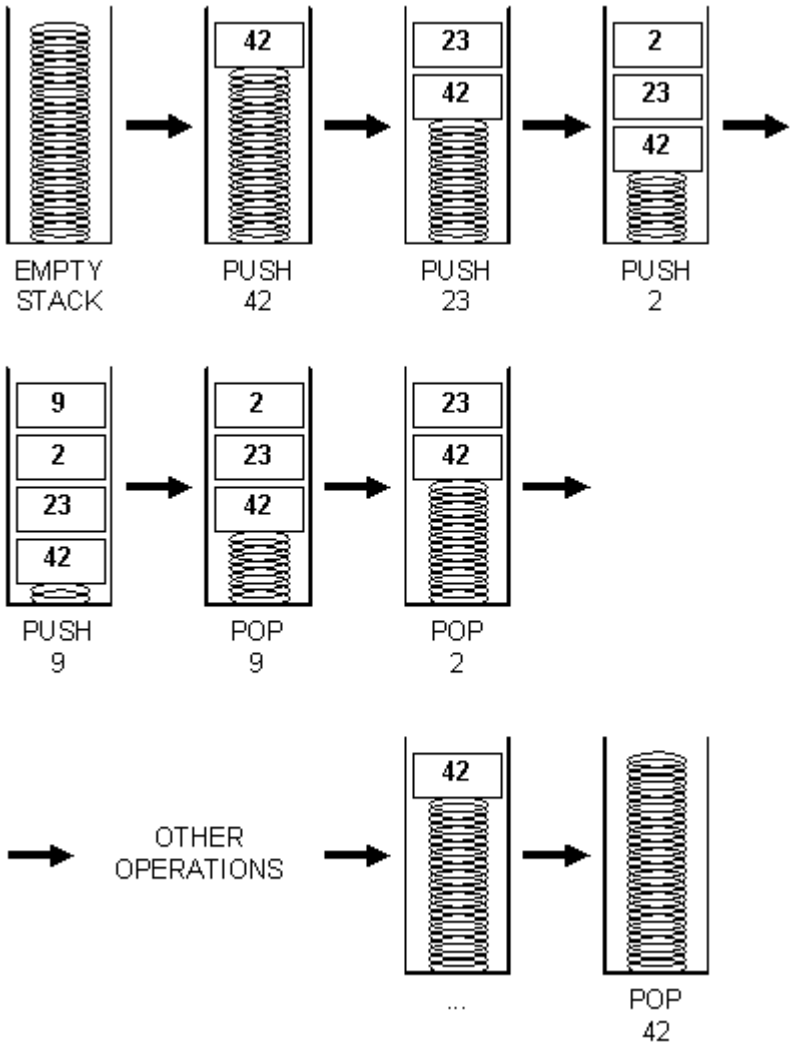
What is the stack? It's a special region of your computer's memory that stores temporary variables created by each function (including the main() function). The stack is a "FILO" (first in, last out) data structure, that is managed and optimized by the CPU quite closely. Every time a function declares a new variable, it is "pushed" onto the stack. Then every time a function exits, **all** of the variables pushed onto the stack by that function, are freed (that is to say, they are deleted). Once a stack variable is freed, that region of memory becomes available for other stack variables.

The advantage of using the stack to store variables, is that memory is managed for you. You don't have to allocate memory by hand, or free it once you don't need it any more. What's more, because the CPU organizes stack memory so efficiently, reading from and writing to stack variables is very fast.

A key to understanding the stack is the notion that **when a function exits**, all of its variables are popped off of the stack (and hence lost forever). Thus stack variables are **local** in nature. This is related to a concept we saw earlier known as **variable scope**, or local vs global variables. A common bug in C programming is attempting to access a variable that was created on the stack inside some function, from a place in your program outside of that function (i.e. after that function has exited).

Another feature of the stack to keep in mind, is that there is a limit (varies with OS) on the size of variables that can be store on the stack. This is not the case for variables allocated on the **heap**.

.



An example of stack operation.

The "Last In" tray is number 9. Thus, the "First Out" tray is also number 9. As customers remove trays from the top of the stack, the first tray removed is tray number 9, and the second is tray number 2. Let us say that at this point more trays were added. These trays would then have to come off the stack before the very first tray we loaded. After any sequence of pushes and pops of the stack of trays, tray 42 would still be on the bottom. The stack would be empty once again only after tray 42 had been popped from the top of the stack.

A useful feature that is included in the CPU of most computers is a stack or last-in first out (LIFO) list. A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The operation a stack can be companied to a stack of trays.

Register Stack:-

A stack can be placed in a portion of a large memory as it can be organized as a collection of a finite number of memory words as register.

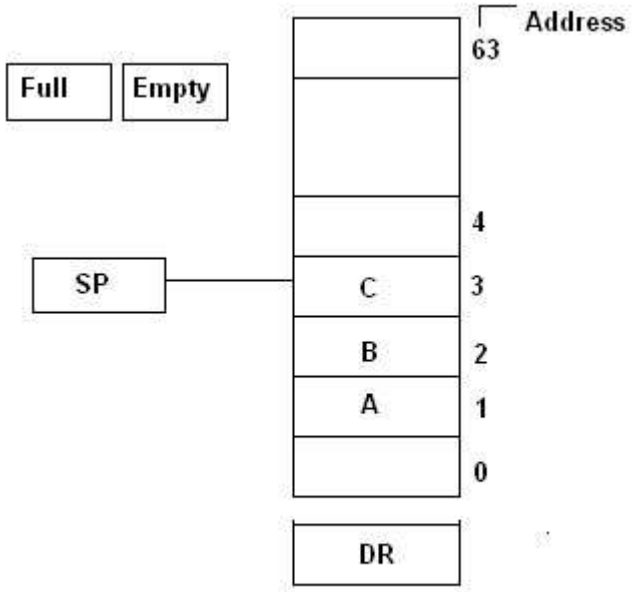


Figure :3 Block Diagram of a 64-word stack

In a 64- word stack, the stack pointer contains 6 bits because 26 = 64.

The one bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty. DR is the data register that holes the binary data to be written into on read out of the stack.

Initially, SP is decide to O, EMTY is set to 1, FULL = 0, so that SP points to the word at address O and the stack is masked empty and not full.

PUSH SP ® SP + 1 increment stack pointer
M [SP] ® DR unit item on top of the Stack
It (SP = 0) then (FULL ® 1) check it stack is full
EMTY ® 0 mask the stack not empty.

POP DR ® [SP] read item trans the top of stack
SP ® SP –1 decrement SP
It (SP = 0) then (EMTY ® 1) check it stack is empty

FULL ® 0

mark the stack not full. A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure X shows the organization of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C in the order. item C is on the top of the stack so that the content of sp is now 3. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next higher location in the stack. Note that item C has read out but not physically removed. This does not matter because when the stack is pushed, a new item is written in its place. In a 64-word stack, the stack pointer contains 6 bits because 26=64.

since SP has only six bits, it cannot exceed a number grater than 63(111111 in binary). When 63 is incremented by 1,
the result is 0 since 111111 + 1 =1000000 in binary,
but SP can accommodate only the six least significant bits. Similarly,
when 000000 is
decremented by 1, the result is 111111. The one bit register Full is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items. DR is the data register that holds the binary data to be written in to or read out of the stack.

Initially, SP is cleared to 0, Emty is set to 1, and Full is cleared to 0, so that SP points to the word at address o and the stack is marked empty and not full. if the stack is not full , a new item is inserted with a push operation. the push operation is implemented with the following sequence of micro-operation.

SP ←SP + 1 (Increment stack pointer)
M(SP) ← DR (Write item on top of the stack)
if (sp=0) then (Full ← 1) (Check if stack is full)
Emty ← 0 (Marked the stack not empty)

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

condition is reached if the top item prior to the last push was in location 63 and after increment SP, the last item stored in location 0. Once an item is stored in location 0, there are no more empty register in the stack. If an item is written in the stack, Obviously the stack can not be empty, so EMTY is cleared to 0.

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

The top item is read from the stack into DR. The stack pointer is then decremented. if its value reaches zero, the stack is empty, so Emty is set to 1. This condition is reached if the item read was in location 1. once this item is read out , SP is decremented and reaches the value 0, which is the initial value of SP. Note that if a pop operation reads the item from location 0 and then SP is decremented, SP changes to 111111, which is equal to decimal 63. In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL=1 or popped when EMTY =1.

COMPUTER ARITHMETIC

A basic operation in all digital computers is the addition or subtraction of two numbers. Arithmetic operations occur at the machine instruction level. They are implemented, along with basic logic functions such as AND,OR, NOT, and exclusive –OR (XOR), in the arithmetic logic unit(ALU) subsystem of the processor. We use logic Circuits to implement arithmetic operations, The time needed to perform and addition operation affects the processor's performance. Multiply and divide operations, which require more complex circuitry than either addition or subtraction operations, also affect performance. We present some of the techniques used in modern computers to perform arithmetic operations at high speed.

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

ADDITION AND SUBTRACTION OF SIGNED NUMBERS

There will arise instances when we need to express numbers that are less than zero. These numbers are called signed numbers and consist of positive(+) and negative(-) numbers. Positive numbers are greater than zero and negative numbers are less than zero.Positive and negative whole numbers are called integers while signed fractions and decimals are called rational numbers. It is not necessary to write the + for a positive number unless you want to draw attention to the fact that it is positive. The negative sign must always in used for a negative number.

A number line for integers continues indefinitely in both the negative and positive directions. Numbers get smaller as we proceed to the left and larger to the right. The opposite of a number is the number the same distance from zero but in the opposite direction.

In order to perform operations with signed numbers, we need to define the absolute value of a number. The absolute value of a number, symbolized by placing the number between 2 vertical bars (l l) is defined to be the distance that number is located from zero on a number line without regard to the direction.

- |3| = 3
- |-5| = 5

When you add two numbers with the same signs add the absolute values, and write the sum (the answer) with the sign of the numbers. If the sign is positive, it is commonly omitted.

5 + 16 = 21
-12 + -15 = -27

Binary Addition Basic Rules for Binary Addition

0+0 = 0 0 plus 0 equals 0
0+1 = 1 0 plus 1 equals 1
1+0 = 1 1 plus 0 equals 1
1+1 = 10 1 plus 1 equals 0
with a carry of 1 (binary 2)

The technique of addition for binary numbers is similar to that for decimal numbers,

except that a 1 is carried to the next column after two 1s are added.

Example: Add the numbers 310 and 110 in binary form.

Solution
The numbers, in binary form, are 11 and 01.
11
01
100
In the right-hand column, 1 + 1 = 0 with a carry of 1 to the next column.
In the next column, 1+0+1 = 0 with a carry of 1 to the next column.
In the left-hand column, 1 + 0 + 0 = 1. Thus, in binary, 11 + 01 = 100 = 410.
Binary Subtraction Basic Rules for Binary Subtraction
0 − 0 = 0 0 minus 0 equals 0
1 − 1 = 0 1 minus 1 equals 0
1 − 0 = 1 1 minus 0 equals 1
10 − 1 = 1 10 minus 1 equals 1

Example: Subtract 310 = 11 from 510 = 101 in binary form.
Solution:-
The subtraction procedure is shown below.
1 0 1 − 0 1 10
1 0 1− 0 1 10
1 0 1− 0 1 11 0
1 0 1− 0 1 10 1 0

Starting from the left, the first array is the subtraction in the right hand column. In the second array, a 1 is borrowed from the third column for the middle column at the top and paid back at the bottom of the third column. The third array is the subtraction 10 − 1 = 1 in the middle column. The final array is the subtraction 1 − 1 = 0 and the final answer is thus 10 = 210.

Introduction About Instruction formats:-

The most common fields found in instruction format are:-

(1) An operation code field that specified the operation to be performed
(2) An address field that designates a memory address or a processor registers.
(3) A mode field that specifies the way the operand or the effective address is determined.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address field in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organization.

(1) Single Accumulator organization ADD X AC ® AC + M [×]
(2) General Register Organization ADD R1, R2, R3 R ® R2 + R3
(3) Stack Organization PUSH X

Three address Instruction

Computer with three addresses instruction format can use each address field to specify either processor register are memory operand.

ADD R1, A, B A1 ® M [A] + M [B]
ADD R2, C, D R2 ® M [C] + M [B] X = (A + B) * (C + A)
MUL X, R1, R2 M [X] R1 * R2

The advantage of the three address formats is that it results in short program when evaluating arithmetic expression. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

Two Address Instruction

Most common in commercial computers. Each address field can specify either a processes register on a memory word.

MOV R1, A R1 ® M [A]
ADD R1, B R1 ® R1 + M [B]
MOV R2, C R2 ® M [C] X = (A + B) * (C + D)
ADD R2, D R2 ® R2 + M [D]
MUL R1, R2 R1 ® R1 * R2
MOV X1 R1 M [X] ® R1

One Address instruction

It used an implied accumulator (AC) register for all data manipulation. For multiplication/division, there is a need for a second register.

```
LOAD    A          AC ® M [A]
ADD     B          AC ® AC + M [B]
STORE T            M [T] ® AC          X = (A +B) × (C + A)
```

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

```
 LOAD    C          AC ® M (C)
ADD      D          AC ® AC + M (D)
ML       T          AC ® AC + M (T)
STORE  X            M [×]® AC
```

Zero – Address Instruction

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

```
PUSH      A      TOS ® A
PUSH      B      TOS ® B
ADD              TOS ®  (A + B)
PUSH      C      TOS ® C
PUSH      D      TOS ® D
ADD              TOS ® (C + D)
MUL              TOS ® (C + D) * (A + B)
POP       X      M [X] TOS
```

CISC Characteristics

A computer with large number of instructions is called complex instruction set computer or CISC. Complex instruction set computer is mostly used in scientific computing applications requiring lots of floating point arithmetic.

1. A large number of instructions - typically from 100 to 250 instructions.
2. Some instructions that perform specialized tasks and are used infrequently.
3. A large variety of addressing modes - typically 5 to 20 different modes.
4. Variable-length instruction formats
5. Instructions that manipulate operands in memory.

RISC Characteristics

A computer with few instructions and simple construction is called reduced instruction set computer or RISC. RISC architecture is simple and efficient. The major characteristics of RISC architecture are,

1. Relatively few instructions
2. Relatively few addressing modes
3. Memory access limited to load and store instructions
4. All operations are done within the registers of the CPU
5. Fixed-length and easily-decoded instruction format.
6. Single cycle instruction execution
7. Hardwired and micro programmed control

Instruction Formats

Opcode, address, addressing mode

Mano uses "address", I use "operand".

- 3-operand (memory-to-memory, register-memory, or load-store)
- 2-operand (memory-to-memory or register-memory)
- 1-address (accumulator-based)
- 0-operand (stack-organized)

Evaluate x = (a + b) * (c + d) in several assembly languages.

VAX (3-operand, memory-to-memory)

x86 (2-operand, register-memory)

MIPS (3-operand, load-store)

Mano (1-operand, accumulator-based)

Introduction About Addressing mode:-

Addressing Modes

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer register as memory words. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction between the operand is activity referenced. Computer use addressing mode technique for the purpose of accommodating one or both of the following provisions.

(1) To give programming versatility to the uses by providing such facilities as pointer to memory, counters for top control, indexing of data, and program relocation.
(2) To reduce the number of bits in the addressing fields of the instruction.

-

The basic operation cycle of the computer

(1) Fetch the instruction from memory
(2) Decode the instruction
(3) Execute the instruction

Program Counter (PC) keeps track of the instruction in the program stored in memory. PC holds the address of the instruction to be executed next and in incremented each time an instruction is fetched from memory.

Addressing Modes: The most common addressing techniques are
• Immediate
• Direct
• Indirect
• Register
• Register Indirect
• Displacement
• Stack
All computer architectures provide more than one of these addressing modes.
The question arises as to how the control unit can determine which addressing mode is being used in a particular instruction. Several approaches are used. Often, different opcodes will use different addressing modes. Also, one or more bits in the instruction format can be used as a mode field. The value of the mode field determines which addressing mode is to be used.
What is the interpretation of effective address. In a system without virtual memory, the effective address will be either a main memory address or a register. In a virtual memory system, the effective address is a virtual address or a register. The actual mapping to a physical address is a function of the paging mechanism and is invisible to the programmer.

Opcode	Mode	Address

Immediate Addressing:

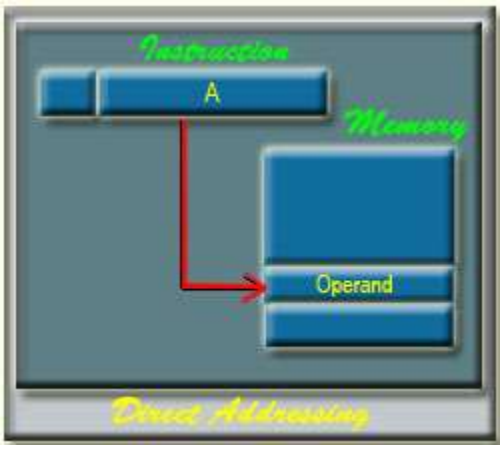
The simplest form of addressing is immediate addressing, in which the operand is actually present in the instruction:

OPERAND = A

This mode can be used to define and use constants or set initial values of variables. The advantage of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the operand. The disadvantage is that the size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the world length.

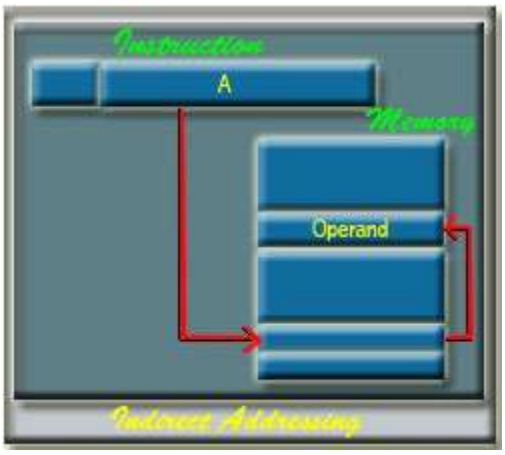
Direct Addressing:

A very simple form of addressing is direct addressing, in which the address field contains the effective address of the operand:
EA = A
It requires only one memory reference and no special calculation.



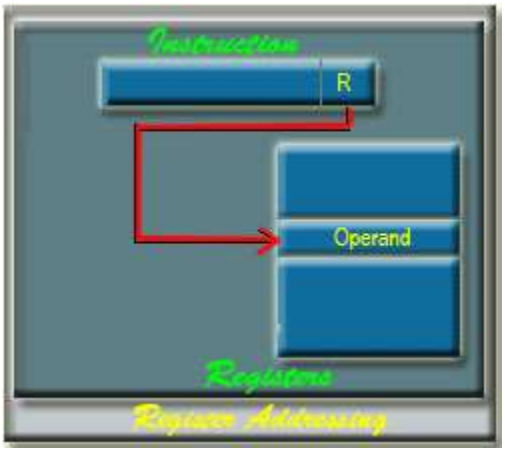
Indirect Addressing:

With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is know as indirect addressing:
EA = (A)



Register Addressing:

Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address:
EA = R
The advantages of register addressing are that only a small address field is needed in the instruction and no memory reference is required. The disadvantage of register addressing is that the address space is very limited.



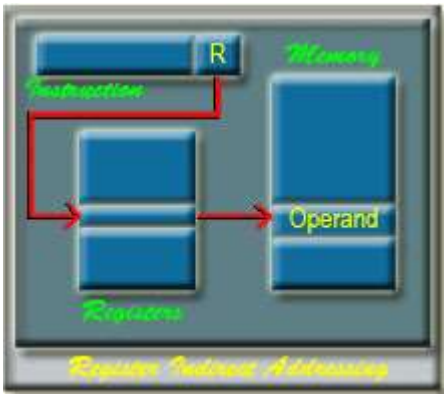
The exact register location of the operand in case of Register Addressing Mode is shown in the Figure 34.4. Here, 'R' indicates a register where the operand is present.

Register Indirect Addressing:

Register indirect addressing is similar to indirect addressing, except that the address field refers to a register instead of a memory location. It requires only one memory reference and no special calculation.

EA = (R)

Register indirect addressing uses one less memory reference than indirect addressing. Because, the first information is available in a register which is nothing but a memory address. From that memory location, we use to get the data or information. In general, register access is much more faster than the memory access.



Displacement Addressing:

A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing, which is broadly categorized as displacement addressing:

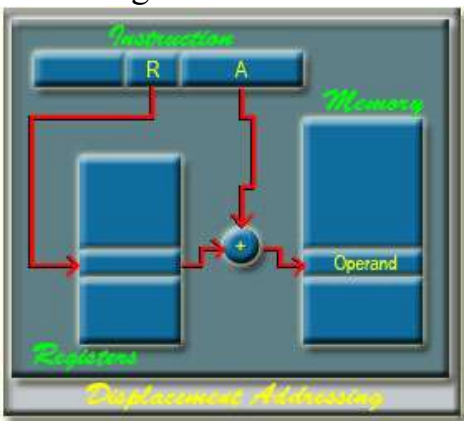
EA = A + (R)

Displacement addressing requires that the instruction have two address fields, at least one of which is explicit. The value contained in one address field (value = A) is used directly. The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.

The general format of Displacement Addressing is shown in the Figure 4.6.

Three of the most common use of displacement addressing are:

• Relative addressing
• Base-register addressing
• Indexing



Relative Addressing:

For relative addressing, the implicitly referenced register is the program counter (PC). That is, the current instruction address is added to the address field to produce the EA. Thus, the effective address is a displacement relative to the address of the instruction.

Base-Register Addressing:

The reference register contains a memory address, and the address field contains a displacement from that address. The register reference may be explicit or implicit. In some implementation, a single segment/base register is employed and is used implicitly. In others, the programmer may choose a register to hold the base address of a segment, and the instruction must reference it explicitly.

Indexing:

The address field references a main memory address, and the reference register contains a positive displacement from that address. In this case also the register reference is sometimes explicit and sometimes implicit. Generally index register are used for iterative tasks, it is typical that there is a need to increment or decrement the index register after each reference to it. Because this is such a common operation, some system will automatically do this as part of the same instruction cycle.

This is known as auto-indexing. We may get two types of auto-indexing: -one is auto-incrementing and the other one is -auto-decrementing.
If certain registers are devoted exclusively to indexing, then auto-indexing can be invoked implicitly and automatically. If general purpose register are used, the auto index operation may need to be signaled by a bit in the instruction.

Auto-indexing using increment can be depicted as follows:
EA = A + (R)
R = (R) + 1
Auto-indexing using decrement can be depicted as follows:
EA = A + (R)
R = (R) – 1

In some machines, both indirect addressing and indexing are provided, and it is possible to employ both in the same instruction. There are two possibilities: The indexing is performed either before or after the indirection.
If indexing is performed after the indirection, it is termed post indexing

EA = (A) + (R)
First, the contents of the address field are used to access a memory location containing an address. This address is then indexed by the register value.

With pre indexing, the indexing is performed before the indirection:
EA = (A + (R)
An address is calculated, the calculated address contains not the operand, but the address of the operand.

Stack Addressing:

A stack is a linear array or list of locations. It is sometimes referred to as a pushdown list or last-in-first-out queue. A stack is a reserved block of locations. Items are appended to the top of the stack so that, at any given time, the block is partially filled. Associated with the stack is a pointer whose value is the address of the top of the stack. The stack pointer is maintained in a register. Thus, references to stack locations in memory are in fact register indirect addresses.
The stack mode of addressing is a form of implied addressing. The machine instructions need not include a memory reference but implicitly operate on the top of

TEXT BOOKS:
1. Carl Hamacher ,(2012). Computer Organization,(5th ed.). McGraw Hill
2. Dos Reis,A.J.,(2009), Assembly Language and Computer Architecture using C++ and JAVA Technology.
3. M.Mories Mano(2013) Computer Systems Architecture,(3rd ed.), Prentice Hall of India
4. Stalings, W., (2010). Computer Organization and Architecture Designing for Performance,(8th ed.), Prentice Hall of India

PART –A(One Mark Questions- Online Examination)

1. An exception conditions in a computer system by an event external to the CPU is called__
   a. interrupt b. halt c. wait d. process
2. Pipelining increases the CPU instruction___
   a. efficiency b. latency c. throughput d. accuracy
3. Which statement is valid about computer program?
   a. it is understood by a computer b. it is understood by programmer
   c. it is understood to use d. all of the above
4. ___is converned with the way the hardware components operate to form computer system.
   a. Computer organization b. Computer design c. Computer architecture d. CAD
5. The unit that operates the CPU bus system and directs the information flow through the registers and ALU is known as
   a) Memory b) processor c) control d) input
6. A _____ is a storage device that stores information that the item stored last is the item retrieved.
   a) Stack b) program counter c) flip-flop d) counter
7. The _____ notation places the operator before the operands.
   a) Infix b) polish c) reverse d) logical
8. Physical memory is divided into sets of finite size called as _____.
   a) Frames b) Pages c) Blocks d) Vectors

PART – B(2 Marks Questions)

1. Define number systems.
2. What are Bus systems?
3. Define CISC.
4. What is meant by interrupts?
5. Give a brief detail about the fixed representation.
6. Describe in detail about the computer registers.

PART - C (6 Marks Questions)

1. Explain in detail the various addressing modes in computer architecture.
2. Distinguish between RISC and CISC processor.
3. Explain in detail the various addressing modes in computer architecture.
4. Distinguish between RISC and CISC processor.
5. Draw and explain CISC Architecture.
6. Discuss in detail about the instruction format with appropriate example.

PART –A(One Mark Questions- Online Examination)

1. The simplest way to determine cache locations in which to store memory blocks is direct in _____
   a) Associate Mapping technique b) Direct mapping technique
   c) Set-Associative Mapping technique d) Indirect mapping technique
2. The timing for all registers in the basic computer is controlled by a
   a) Memory b) processor c) clock generator d) control unit
3. The addressing mode which makes use of in-direction pointers is known as _____
   a) Indirect addressing mode b) Index addressing mode
   c) Relative addressing mode d) Offset addressing mode
4. A _____ is a storage device that stores information that the item stored last is the item retrieved.
   a) Stack b) program counter c) flip-flop d) counter
5. Memory access in RISC architecture is limited to instructions
   a. CALL and RET b. PUSH and POP c. STA and LDA d. MOV and JMP
6. A microprogram written as string of 0's and 1's is a____
   a. symbolic micro-instructions b. binary micro-instructions
   c. symbolic micro-instruction d. binary microprogram
7. The unit that operates the CPU bus system and directs the information flow through the registers and ALU is known as
   a) Memory b) processor c) control d) input
8. A _____ is a storage device that stores information that the item stored last is the item retrieved.
   a) Stack b) program counter c) flip-flop d) counter

PART – B(2 Marks Questions)

1. List the phases of the instruction cycle.
2. Define RISC.
3. What is meant by peripheral devices? Give one example.
4. Define Auxiliary Memory.
5. What are the associative memory?

PART - C (6 Marks Questions)

1. Define ROM memory? Explain in detail various ROM memory.
2. What is RAM memory? Explain in detail the various RAM memory.
3. Explain in detail the associative memory with relevant diagram.
4. Describe in detail the direct memory access with appropriate diagram.
5. Describe in detail about IOP organization.
6. Describe the data transfer method using DMA.
7. Write a note on i) Associative Memory ii) Cache Memory.
8. Draw and Explain the block diagram of Direct Memory Access.
9. What is cache memory? Discuss in details the organization of cache memory.

UNIT – V CACHE MEMORY

Cache Memory

Cache memory, also called CPU memory, is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly with the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU.

The basic purpose of cache memory is to store program instructions that are frequently re-referenced by software during operation. Fast access to these instructions increases the overall speed of the software program.

As the microprocessor processes data, it looks first in the cache memory; if it finds the instructions there (from a previous reading of data), it does not have to do a more time-consuming reading of data from larger memory or other data storage devices.

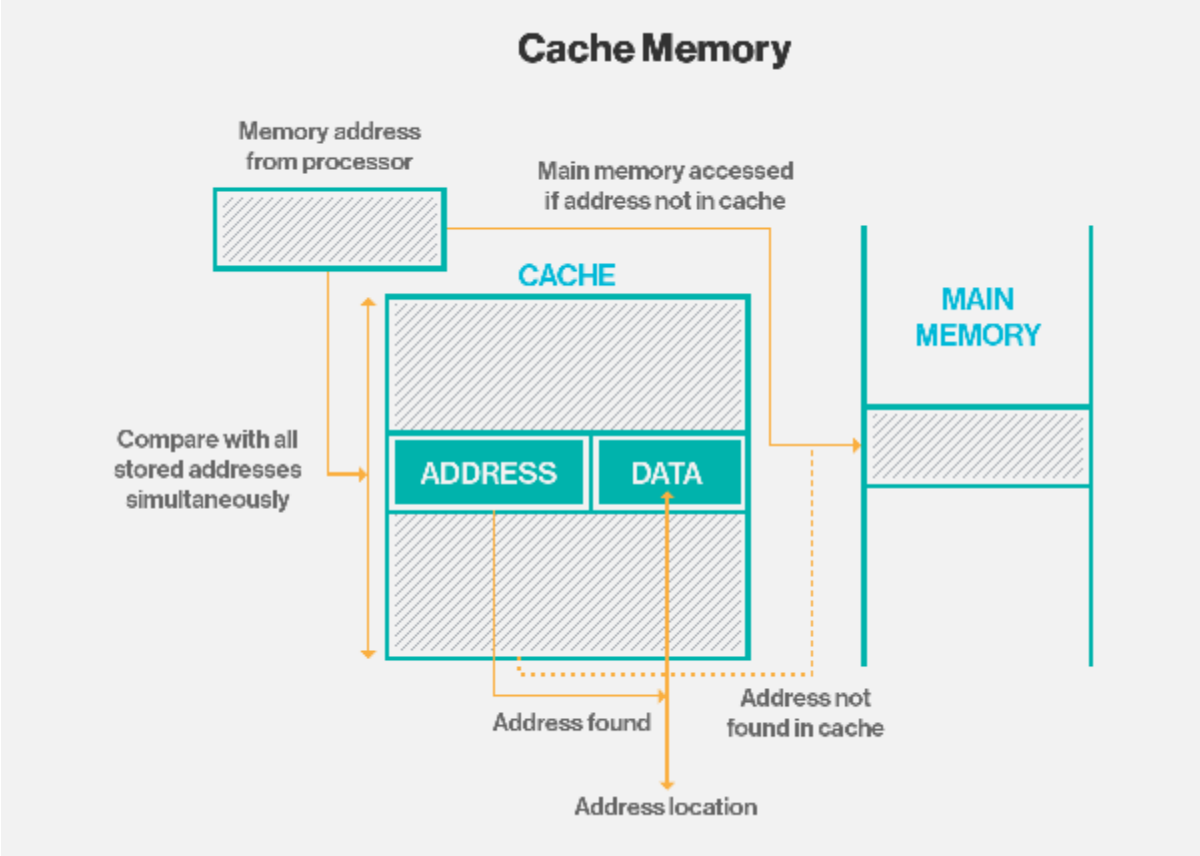
Most programs use very few resources once they have been opened and operated for a time, mainly because frequently re-referenced instructions tend to be cached. This explains why measurements ofsystem performance in computers with slower processors but larger caches tend to be faster than measurements of system performance in computers with faster processors but more limited cache space.

Multi-tier or multilevel caching has become popular in server and desktop architectures, with different levels providing greater efficiency through managed tiering. Simply put, the less frequently access is made to certain data or instructions, the lower down the cache level the data or instructions are written.

Cache memory levels explained

Cache memory is fast and expensive. Traditionally, it is categorized as "levels" that describe its closeness and accessibility to the microprocessor:

- Level 1 (L1) cache is extremely fast but relatively small, and is usually embedded in the processor chip (CPU).
- Level 2 (L2) cache is often more capacious than L1; it may be located on the CPU or on a separate chip or coprocessor with a high-speed alternative system bus interconnecting the cache to the CPU, so as not to be slowed by traffic on the main system bus.
- Level 3 (L3) cache is typically specialized memory that works to improve the performance of L1 and L2. It can be significantly slower than L1 or L2, but is usually double the speed of RAM. In the case of multicore processors, each core may have its own dedicated L1 and L2 cache, but share a common L3 cache. When an instruction is referenced in the L3 cache, it is typically elevated to a higher tier cache.



Memory cache configurations

Caching configurations continue to evolve, but memory cache traditionally works under three differentconfigurations:

- Direct mapping, in which each block is mapped to exactly one cache location. Conceptually, this is like rows in a table with three columns: the data block or cache line that contains the actual data fetched and stored, a tag that contains all or part of the address of the fetched data, and a flag bitthat connotes the presence of a valid bit of data in the row entry.
- Fully associative mapping is similar to direct mapping in structure, but allows a block to be mapped to any cache location rather than to a pre-specified cache location (as is the case with direct mapping).
- Set associative mapping can be viewed as a compromise between direct mapping and fully associative mapping in which each block is mapped to a subset of cache locations. It is sometimes called N-way set associative mapping, which provides for a location in main memory to be cached to any of "N" locations in the L1 cache.

Specialized caches

In addition to instruction and data caches, there are other caches designed to provide specialized functions in a system. By some definitions, the L3 cache is a specialized cache because of its shared design. Other definitions separate instruction caching from data caching, referring to each as a specialized cache.

Other specialized memory caches include the translation lookaside buffer (TLB) whose function is to record virtual address to physical address translations.

Still other caches are not, technically speaking, memory caches at all. Disk caches, for example, may leverage RAM or flash memory to provide much the same kind of data caching as memory caches do with CPU instructions. If data is frequently accessed from

disk, it is cached into DRAM or flash-basedsilicon storage technology for faster access and response.

In the video below, Dennis Martin, founder and president of Demartek LLC, explains the pros and cons of using solid-state drives as cache and as primary storage.

Associative Memory

- Also called as Content-addressable memory (CAM), associative storage, or associative array
- Content-addressed or associative memory refers to a memory organization in which the memory is accessed by its content (as opposed to an explicit address).
- It is a special type of computer memory used in certain very high speed searching applications.
- In standard computer memory (random access memory or RAM) the user supplies a memory address and the RAM returns the data word stored at that address.
- In CAM the user supplies a data word and then CAM searches its entire memory to see if that data word is stored anywhere in it. If the data word is found, the CAM returns a list of one or more storage addresses where the word was found.
- CAM is designed to search its entire memory in a single operation.
- It is much faster than RAM in virtually all search applications.
- An associative memory is more expensive than RAM, as each cell must have storage capability as well as logic circuits for matching its content with an external argument.
- Associative memories are used in applications where the search time is very critical and short.
- Associative memories are expensive compared to RAMs because of the add logic associated with each cell.

Input Output Interface:

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

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

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

Need for I/O interface

1. Peripherals are electromechanical devices.But CPU and Memory are electronic devices. Therefore conversion of signal values may be required.

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

I/O versus Memory Bus

The processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address, and read/write control lines. There are three ways that computer buses can be used to communicate with memory and I/O:

1. Use two separate buses, one for memory and the other for I/O.
2. Use one common bus for both memory and I/O but have separate control lines for each.
3. Use one common bus for memory and VO with common control lines.

In the first method, the computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O. This is done in computers that provide a separate I/O processor (IOP) in addition to the central processing unit (CPU). The memory communicates with both the CPU and the IOP through a memory bus. The IOP communicates also with the input and output devices through a separate I/O bus with its own address, data and control lines. The purpose of the IOP is to provide an independent pathway for the transfer of information between external devices and internal memory. The I/O processor is sometimes called a data channel.

Isolated versus Memory Mapped I/O

When Memory and I/O devices are interfaced to a processor either an isolated I/O scheme or memory mapped I/O scheme is used.

Isolated I/O:

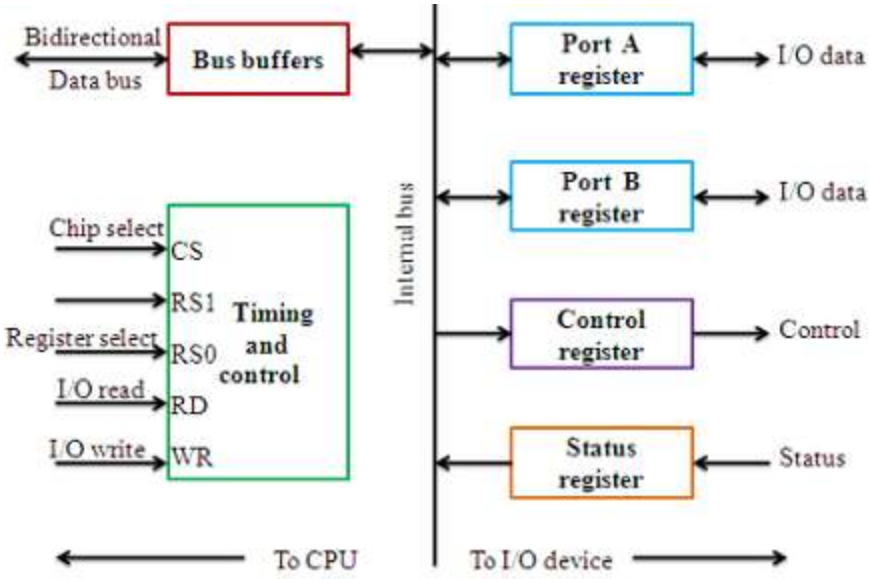
The isolated I/O configuration separates all I/O interface addresses from the memory addresses. In the isolated I/O configuration, the CPU has distinct input and output instructions. In isolated I/O configuration the memory address and I/O address have its own address space. If the address of interface registers are placed on the address lines the I/O read or I/O write control lines are enabled. If the memory address is placed on the address lines the memory read and memory write control lines are enabled.

Memory - Mapped I/O:

In this configuration same address space is used for both memory and I/O. There are no specific I/O instructions. It allows the computer to used the same instructions for both I/O transfers and memory transfers. Some instructions are memory reference instructions and others are I/O reference. They are only one set of read/write control signals.

Example of I/O Interface

An example of an I/O interface unit is shown in figure. It consists of two data registers called ports, a control register, a status register, bus buffers and timing and control circuits.



The four registers communicate directly with the I/O device attached to the interface. The I/O data to and from the device can be transferred into either port A or port B. Port A may be defined as an input port and port B may be defined as an output port. The output device such as magnetic disk transfers data in both directions. So bidirectional data bus is used. CPU gives control information to control register. The bits in the status register are used for status conditions. It is also used for recording errors that may occur during the data transfer. The bus buffers use the bidirectional data bus to communicate with the CPU. A timing and control circuit is used to detect the address assigned to the bus buffers.

CS	RS1	RS0	Register selected
0	X	X	None: data bus in high-impedance
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

TEXT BOOKS:

1. Carl Hamacher ,(2012). Computer Organization,(5th ed.). McGraw Hill
2. Dos Reis,A.J.,(2009), Assembly Language and Computer Architecture using C++ and JAVA Technology.
3. M.Mories Mano(2013) Computer Systems Architecture,(3rd ed.), Prentice Hall of India
4. Stalings, W., (2010). Computer Organization and Architecture Designing for Performance,(8th ed.), Prentice Hall of India

B.Sc., CS/ BCA / B.Sc. IT / Comp. Technology Op-569

Reg. No.....

[13CAU102/13CSU102/13ITU102]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2013 onwards)

BCA & B.Sc. DEGREE EXAMINATION, JANUARY 2016

First Semester

COMPUTER APPLICATIONS/COMPUTER SCIENCE/

INFORMATION TECHNOLOGY

DIGITAL ELECTRONICS

Time: 3 hours Maximum : 60 marks

PART – A (10 x 2 = 20 Marks)
Answer any TEN Questions

1. Give examples of Decimal Numbers and Hexa decimal number
2. What is an ASCII Code?
3. What is gray Code?
4. Draw OR Gate
5. Draw the circuit diagram for NOT gate by using NOR
6. Give the truth table of AND Gate
7. What is comparator?
8. Explain encoder with example.
9. Define Parity Generator
10. What is sequential logic circuit?
11. Draw RS Flip flop
12. Give the truth table of D flip flop
13. What is a register?
14. What is counter?
15. List out the various types of A/D converter

PART B (5 X8= 40 Marks)
Answer ALL the Questions

16. a. Convert the following:
(i) $(127)_8 = (\quad)_2$ (ii) $(1011.1011)_2 = (\quad)_{16}$
(ii) Explain gray code with example.
Or
b. i) Convert Binary to Decimal
i) 11011 ii) 11.11 iii) 0.01011 iv) 111101

ii) Write a short notes on 1. ASCII 2. Error detection

17. a. i) State and Explain De-Morgan's Theorem.
ii) Draw the circuit diagram for basic gates by using NAND gate
Or
b. Explain the following:
(i) Associative law (ii) Commutative law
c. Solve using Karnaugh map for the function
$F(A,B,C,D) = \sum(0,1,2,4,8,10,11,12,14,15)$.

18. a. How will you convert a D flip-flop into JK flip-flop?
Or
b. Briefly explain about encoder and decoder with neat circuit diagram

19. a. Explain briefly on Clocked JK Flip Flop
Or
b. i) Give the difference between synchronous counter and asynchronous counter
ii) Explain Up/Down Counters with neat diagram

20. a. (i) Briefly explain the circuit diagram of SAC A/D converter.
(ii) Explain D/A resistor network with circuit diagram and find the output analog Voltage
Or
b. Explain types of shift register and Application of shift registers

20. If the main memory is of 8K bytes and the cache memory is of 2K words. It uses associative mapping. Then each word of cache memory shall be
a. 11 bits b. 21 bits c. 16 bits d. 20 bits

PART B (5 x 2 = 10 Marks) (2 ½ Hours)
Answer ALL the Questions

21. Define combinational circuits.
22. Convert the binary number $(1011101010000101)_2$ into octel number.
23. Write short notes on interrupt.
24. What is a machine language?
25. What is cache memory?

PART C (5 x 6 = 30 Marks)
Answer ALL the Questions

26. a. Discuss in detail shift registers with an example.
(Or)
b. Explain in detail 8:1 multiplexer with neat diagram.

27. a. Explain adder and subtractor with examples.
(Or)
b. Convert the following numbers, i. $(1246)_8$=($)_2$ ii. $(ABCF7)_{16}$ = ($)_2$

28. a. Discuss in detail about the interconnection structures.
(Or)
b. Explain the functions of instruction cycle with neat diagram.

29. a. Explain in detail the various addressing modes in computer architecture.
(Or)
b. Distinguish between RISC and CISC processor.

30. a. Explain in detail the associative memory with relevant diagram.
(Or)
b. Describe in detail the direct memory access with appropriate diagram.

Reg. No.....

[16CAU102/16CSU102/16ITU102/16CTU102]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2016 onwards)

BCA., B.Sc., DEGREE EXAMINATION, JANUARY 2017

First Semester

COMPUTER APPLICATIONS/COMPUTER SCIENCE/INFORMATION TECHNOLOGY/COMPUTER TECHNOLOGY

COMPUTER SYSTEM ARCHITECTURE

Time: 3 hours Maximum : 60 marks

PART – A (20 x 1 = 20 Marks) (30 Minutes)
Answer ALL the Questions

1. Group of Flip-Flop is called___.
a. register b. counter c. shift d. rotator

2. Combinational circuits have ____
a. no feed back b. feedback c. past feedback d. present feedback

3. Boolean Algebra A'. A=____
a. 0 b. 1 c. A d. A'

4. Sequential circuits have
a. no feed back b. feedback c. past feedback d. present feedback

5. What is the base number of decimal number
a. 2 b. 4 c. 8 d.10

6. What is the one's complement of $(1000110)_2$
a. 0111001 b. 0111101 c. 1010000 d. 0001110

7. Convert $(127)_{10}$ to binary number $(\ )_2$
a. 1111111 b.100101010 c. 101010101 d. 010101010

8. Signed number is used to indicate ____ numbers
a. negative b. positive c. negative and positive d. zero

9. An address in main memory is called
a. Physical address b. Logical address c. Memory address
d. Word address

10. Status bit is also called
a. Binary bit b. Flag bit c. Signed bit d. unsigned bit

11. A pipeline is like .....
a. an automobile assembly line b. house pipeline c. both a and b
d. a gas line

12. Data hazards occur when .....
a. Greater performance loss
b. Pipeline changes the order of read/write access to operands
c. Some functional unit is not fully pipelined d. Machine size is limited

13. In a vectored interrupt.
a. The branch address is assigned to a fixed location in memory.
b. The interrupting source supplies the branch information to the processor through an interrupt vector.
c. The branch address is obtained from a register in the processor
d. The branch address is obtained from a memory in the processor

14. The circuit used to store one bit of data is known as
a. Encoder b. OR gate c. Flip Flop d. Decoder

15. Cache memory acts between
a. CPU and RAM b. RAM and ROM c. CPU and Hard Disk d. ROM

16. Write Through technique is used in which memory for updating the data
a. Virtual memory b. Main memory c. Auxiliary memory
d. Cache memory

17. Generally Dynamic RAM is used as main memory in a computer system as it
a. Consumes less power b. has higher speed c. has lower cell density
d. needs refreshing circuitry

18. In a program using subroutine call instruction, it is necessary
a. initialize program counter b. Clear the accumulator
c. Reset the microprocessor d. Clear the instruction register

19. A Stack-organised Computer uses instruction of
a. Indirect addressing b. Two-addressing c. Zero addressing
d. Index addressing

Reg. No.....

[12CSU102/12ITU102/12CAU102]

KARPAGAM UNIVERSITY

(Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2012 onwards)

B.Sc. & BCA DEGREE EXAMINATION, NOVEMBER 2014

First Semester

COMPUTER SCIENCE /INFORMATION TECHNOLOGY/ COMPUTER APPLICATIONS

DIGITAL ELECTRONICS

Time: 3 hours Maximum : 100 marks

PART – A (15 x 2 = 30 Marks)
Answer ALL the Questions

1. Convert the following Hexadecimal numbers into its equivalent binary number?
   (i) 2A5 (ii) ADC
2. Convert the following gray to binary number.
   (i) 110011 (ii) 11110011
3. Perform binary addition for a signed number of +95 and -23.
4. Define Duality theorem.
5. 5 Give the logic symbol and truth table of EX-OR gate.
6. Define Associative law and Commutative law
7. Define K-map. Give an example.
8. What is an Encoder?
9. Define combinational logic.
10. Define sequential logic.
11. Define Synchronous counter.
12. Write short notes on Shift registers.
13. Give the logic diagram of Simultaneous A/D conversion
14. What is the advantage of R/2R ladder DACs over those that use binary- weighted Register?
15. Give the logic diagram of frequency measurement.

PART B (5 X 14= 70 Marks)
Answer ALL the Questions

16. a) (i) Convert the decimal number 82.67 to its binary, hexadecimal and octal Equivalents.
    (ii) Add 20 and (-15) using 2's complement.
    Or
    b) Perform the following subtractions using 2's complement method.
    (i) 01000 – 01001 (ii) 01100 – 00011 (iii) 0011.1001 – 0001.1110

17. a) simplify the following SOP equation using the Boolean algebra and realize the simplified equation $Y = \overline{A}.\overline{B}.\overline{D} + A.\overline{C}.\overline{D} + \overline{A}.B.\overline{C} + A.B.\overline{C}.D + A.\overline{B}.C.\overline{D}$
    Or
    b) Simplify the Boolean function using k-map method
    $F(A,B,C,D,E) = \sum m(0,2,4,6,9,13,21,23,25,29,31)$

18. a) What is a full adder? Explain a full-adder with the help of truth-table and logic diagram.
    Or
    b) Using a suitable logic diagram, explain the working of a 1-to-16 de multiplexer.

19. a) Design a mod-10 Synchronous up counter.
    Or
    b) Describe the operation of parallel in parallel out (PIPO) shift register.

20. a) Describe the operation of voltage to frequency ADC.
    Or
    b) With the help of a neat diagram, explain the working of a weighted-resistor D/A converter.

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2015 onwards)

BCA, B.Sc. DEGREE EXAMINATION, NOVEMBER 2016

First Semester

COMPUTER APPLICATIONS/COMPUTER SCIENCE/ INFORMATION TECHNOLOGY/ COMPUTER TECHNOLOGY

DIGITAL ELECTRONICS

Time: 3 hours Maximum : 60 marks

PART – A (20 x 1 = 20 Marks)
Answer ALL the Questions

1. A 16-bit value is known as ____ in a digital systems
   a. Bit b. Byte c. Nibble d. Word

2. The 1's complement value of 1010 is____
   a. 1010 b. 1100 c. 101 d. 1011

3. The sum of two binary numbers (100)2 and (10)2is____
   a. 111 b. 101 c. 110 d. 10

4. If total number of 1's is even is a binary number ,then it is called_____ error detection
   a. Odd parity b. Even parity c. Hollerith code d. Hamming code

5. The minimal form of the Boolean Expression F=(x+y)+x'y' is
   a. 1 b. 0 c. x+y d. x'y'

6. The output of a NOT gate is HIGH when _____.
   a. the input is LOW b. the input is HIGH
   c. power is applied to the gate's IC d. power is removed from the gate's IC

7. Which statement below best describes a Karnaugh map?
   a. Variable complements can be eliminated by using Karnaugh maps.
   b. It is simply a rearranged truth table.

c. A Karnaugh map can be used to replace Boolean rules.

8. The basic logic gate whose output is the complement of the input is the:
   a. INVERTER gate b. comparator c. OR gate d. AND gate

9. A demultiplexer has one data input, m control inputs and n outputs, then
   a. $2^n = m$ b. $2^m = n$ c. $n^m = 2$ d. $m^n = 2$

10. A PAL is a
    a. MSI device b. LSI device c. VLSI device d. SSI device

11. The PLA consists of
    a. Programmable AND-array and fixed OR-array
    b. Fixed AND-array and fixed OR-array
    c. Fixed AND-array and Programmable OR-array
    d. Programmable OR-array and Programmable AND-array

12. The demultiplexer is basically a combinational logic circuit to perform the operation
    a. AND – AND b. OR-OR c. AND-OR d. OR-AND

13. Following Flip Flop is used to eliminate race around problem
    a. R-S Flip Flop b. Master Slave J-K Flip Flop
    c. J-K Flip Flop d. D Flip Flop

14. The decoding glitches as far more likely to occur in the case of
    a. Ripple counters b. Parallel Counters
    c. Johnson Counters d. Ring Counters

15. The Table which shows the necessary levels at D inputs to produce every possible flip-flop state transition is called
    a. Truth Table of D Flip-flop b. Excitation Table of D Flip-flop
    c. Excitation Table of Mod-N counters using D Flip-flop d. Both A and B

16. A Four bit Counter
    a. has a modulus of 4 b. cannot have a modulus greater than 16
    c. has a modulus that is less than or equal to 16 d. Both B and C are true

17. Which of the following are the drawbacks of flash type ADC?
    i) slowest ADC
    ii) number of comparator almost doubles for each added bit
    iii) most expensive
    iv) for larger value of bits, more complex is the priority encoder
    a. only I b. only iv c. only i,ii and iii d. only ii and iv

Reg. No.....

[14CAU102/14CSU102/14ITU102/14CTU102]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2014 onwards)

BCA & B.Sc. DEGREE EXAMINATION, NOVEMBER 2016

First Semester

COMPUTER APPLICATIONS / COMPUTER SCIENCE/ INFORMATION TECHNOLOGY / COMPUTER TECHNOLOGY

DIGITAL ELECTRONICS AND COMPUTER SYSTEM ARCHITECTURE

Time: 3 hours Maximum : 60 marks

PART – A (10 x 2 = 20 Marks)
Answer any TEN Questions

1. Convert the following no from octal to decimal
   i. 557 ii. 1024
2. State the distributive property of Boolean algebra?
3. Define Minterm
4. Draw the Half subtractor circuits.
5. List out the application of Multiplexers
6. Draw the combination logic circuit for given equation (A+B) (B+C) (C+A)
7. What is data register?
8. Define combinational logic.
9. What are the major component types at the register level?
10. Define latency.
11. What are the functions of control unit?
12. Define parallel processing.
13. What are the steps taken when an interrupt occurs?
14. What is micro programmed control?
15. What is programmed I/O?

PART B (5 X 8= 40 Marks)
Answer ALL the Questions

16. a.. Explain weighted and non-weighted codes with examples.

(or)

b. Convert the Following
   i. $(9A.A)_{16}$ ii. $(62)_8$ iii. $(1101)_2$ iv. $(1011010)_2$ v. $(3FC.8)_{16}$

17. a. Design a combinational circuit that performs the arithmetic sum of three input bit.

(or)

b. Design and explain a comparator with neat diagram.

18. a. With a neat diagram explain the operation of 4 bit SISO (serial in –Serial out) register .Give its truth table.

(or)

b. With a neat diagram explain the clocked RS flip-flop with truth table.

19. a. Explain the pipeline processing concepts in detail.

(or)

b. Explain in detail about instruction set used in computer system.

20. **Compulsory : -**

Design sequential circuit based ALU and analysis its operation in detail

---

Reg. No.....

[16CAU102/16CSU102/16ITU102/16CTU102]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2016 onwards)

BCA., B.Sc., DEGREE EXAMINATION, NOVEMBER 2016

First Semester

COMPUTER APPLICATIONS/COMPUTER SCIENCE/INFORMATION TECHNOLOGY/COMPUTER TECHNOLOGY

COMPUTER SYSTEM ARCHITECTURE

Time: 3 hours Maximum : 60 marks

PART – A (20 x 1 = 20 Marks) (30 Minutes)
(Question Nos. 1 to 20 Online Examinations)

PART B (5 x 2 = 10 Marks) (2 ½ Hours)
Answer ALL the Questions

21. Define flip-flops?
22. What is decimal?
23. Explain instruction cycle?
24. What is control word?
25. Define associative memory?

PART C (5 x 6 = 30 Marks)
Answer ALL the Questions

26. a) Explain in details about encoder?
Or
b) Write down registers with example?

27. a) Discuss about integer representation?
Or
b) Briefly explain about complements with example?

28. a) Discuss about computer registers?
Or
b) Briefly explain about input-output instructions?

29. a) Briefly explain about register stack?
Or
b) Explain three-address instructions?

30. a) Write down DMA controller with block diagram?
Or
b) Explain about write operation in associative memory?

---

16. The Maximum modulo number that can be obtained by a ripple counter using ten flip-flops is
a. 16 b. 32 c. 5 d. 1024

17. The time required for the conversion of analog signal into a digital equivalent is called _____ time.
a. aperture b. acquisition c. conversion d. settling

18. The main disadvantage of binary weighted resistor method is _____
a. word length of binary word is small b. more than c. less resolutions
d. wide range of resistor values

19. JFET is used as a shunt switch. When V_{GS}=0, JFET acts as _____ switch and the output V0= _____.
a. open, zero b. open, Vin c. closed, Vin d. closed, zero

20. In dual slope ADC, the unknown voltage and the reference voltages are converted into an equivalent _____using an integrator.
a. time period b. frequency c. current d. digital

PART B (5 x 8 = 40 Marks) (2 ½ Hours)
Answer ALL the Questions

21. a) Convert the following:
i. $(101)_{10}$ = $(?)_2$ ii. $(149)_{10}$ = $(?)_2$ iii. $(9A.A)_{16}$ =$(?)_2$
iv. Convert Gray to binary 110110111
Or
b) Perform the Arithmetic operation
i. 11101+111 ii. 1111-101 iii. 11110
Convert the given no into 1's and 2's complement(11101010)

22. a) Simplify the expression $y=\sum_m(0,1,2,3,4,5,,11,12,14,15)$.
Or
b) State and explain Demorgan's theorem.

23. a) Describe the adder circuits with neat block diagram.
Or
b) Explain in detail about the 4-bit parallel Adder/Subtractor

24. a) Draw and explain the 4 bit ring counter using D Flip Flop.
Or
b) Explain JK flip flop with the master slave with neat diagram

25. a) Describe the Successive approximation A/D conversion in detail.
Or
b) Explain 4 bit weighted resistor type D/A converter in detail.

[15CAU102/15CSU102/15ITU102/15CTU102]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2015 onwards)

BCA, B.Sc. DEGREE EXAMINATION, JANUARY 2016

First Semester

COMPUTER APPLICATIONS/COMPUTER SCIENCE/ INFORMATION TECHNOLOGY/ COMPUTER TECHNOLOGY

DIGITAL ELECTRONICS

Time: 3 hours Maximum : 60 marks

PART – A (20 x 1 = 20 Marks) (30 Minutes)
Answer ALL the Questions

1. The single digit value in digital system is known as_____
   a. Bit b. Byte c. Nibble d. Word

2. The octal value of a decimal number 0.23 is___
   a. 0.1432 b. 0.732 c. 0.1656 d. 0.1414

3. What is the weight of binary digit 1001 using 8421 code____
   a. 6 b. 9 c. 13 d. 15

4. The ____method is used to detect double bit errors
   a. Parity bit b. Hamming code c. Hollerith code d. Check Sum

5. When used with an IC, what does the term "QUAD" indicate?
   a. 4 circuits b. 2 circuits c. 8 circuits d. 6 circuits

6. Which of the following logical operations is represented by the + sign in Boolean algebra?
   a. inversion b. AND c. OR d. complementation

7. Which of the following gates has the exact inverse output of the OR gate for all possible input combinations?
   a. AND b. NAND c. NOR d. NOT

a. the way we OR or A[illegible]
   same
b. we can group variables in an AND or in an OR any way we want
c. the factoring of Boolean expressions requires the multiplication of produ
   that contain like variables
d. an expression can be expanded by multiplying term by term just the same
   ordinary algebra

9. A Decoder is nothing but a Demutilplexer without
   a . Control inputs b. Data input c. Enable input d. Both A and

10. The Basic difference between a CPLD and an FPGA is CPLD architecture
    comprises smaller amount of Programmable sum of Products logic arrays
    a. FPGA architecture is dominated by Programmable interconnectors and
       configurable logic blocks
    b. CPLD architecture comprises smaller number of configurable logic bloc
       while
       FPGA architecture is dominated by large number of configurable logic b
    c. CPLD architecture comprises Programmable interconnectors and configu
       logic blocks while FPGA architecture is dominated by smaller number of
       Programmable sum of Products logic array
    d. Both A and B

11. A DEMUX is a
    a. VLSI device b. MSI device c. SSI device d. LSI device

12. In a 2-to-1 multiplexer the no. of control inputs will be
    a. 2 b. 3 c. 1 d. 4

13. Shifting a binary data to the left by one bit position using Shift Left register, amount to
    a. Addition of 2 b. Multiplication by 2 c. Subtraction of 2
    d. Division by 2

14. A Five bit Counter
    a. has a modulus of 5 b. cannot have a modulus greater than 32
    c. has a modulus that is less than or equal to 32 d. Both B and C are true

15. The Table which shows the necessary levels at T inputs to produce every possib
    flip-flop state transition is called
    a. Truth Table of T Flip-flop b. Excitation Table of T Flip-flop
    c. Excitation Table of Mod-N Counter using T Flip-flop d. Both A and B

Reg. No.....

[14CAU102/14CSU102/14ITU102/14CTU102]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2014 onwards)

BCA & B.Sc. DEGREE EXAMINATION, JANUARY 2016
First Semester

COMPUTER APPLICATIONS / COMPUTER SCIENCE/ INFORMATION TECHNOLOGY / COMPUTER TECHNOLOGY
DIGITAL ELECTRONICS AND COMPUTER SYSTEM ARCHITECTURE

Time: 3 hours Maximum : 60 marks

PART – A (10 x 2 = 20 Marks)
Answer any TEN Questions

1. Convert the following binary no to the decimal
   i. 10001101 ii. 10111.1011
2. Define Associative law
3. What are the applications of octal number system?
4. Draw the Half Adder circuit.
5. What is meant by Decoder?
6. What are the types of code converter?
7. What are the major computer design levels?
8. What is shift register?
9. What is ripple counter?
10. Define computer architecture.
11. What is meant by cache memory?
12. What are the uses of interrupts?
13. What is DMA?
14. What is meant by data path element?
15. What is meant by virtual memory?

PART B (5 X 8= 40 Marks)
Answer ALL the Questions

16. a. Convert the following:
   (i) $(127)_8 = (\quad)_2$ (ii) $(1011.1011)_2 = (\quad)_{16}$
   (iii) $(29.25)_{10} = (\quad)_8$ (iv) $(FF)_{16} = (\quad)_{10}$

   Or

   b. Explain weighted and non-weighted codes with examples.

Q9-569

17. a. What is meant by a priority encoder? Explain with a neat diagram the functi of an encoder?

   Or

   b. Why multiplexer is called as data selector? Explain with a circuit diagram.

18. a. With a neat diagram explain the operation of 4 bit left shift register Give its truth table.

   Or

   b. Design a 3 bit binary counter and write the truth table and output waveforms.

19. a. Explain the combinational ALU design in detail?

   Or

   b. Explain in detail about instruction set used in computer system.

20. **Compulsory : -**

   Analysis the memory device characteristics with the various parameters associated with it

---