CS3351-DIGITAL PRINCIPLES AND COMPUTER ORGANIZATION

UNITICOMBINATIONALCIRCUITS:

-Combinational Circuits

– Karnaugh Map - Analysis and Design Procedures

– Binary Adder – Subtractor

– Decimal Adder

- Magnitude Comparator

– Decoder

– Encoder

– Multiplexers

- Demultiplexers

INTRODUCTION:

Thedigitalsystemconsistsoftwotypesofcircuits,namely

(i) Combinationalcircuits

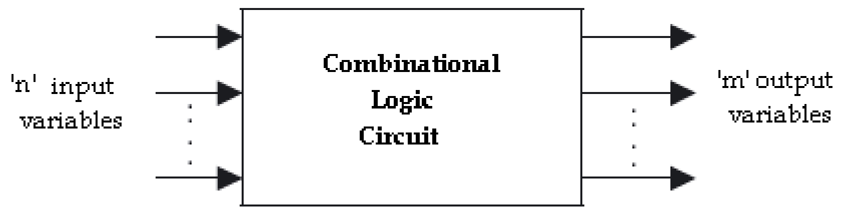
(ii) Sequentialcircuits

Combinationalcircuitconsistsoflogicgateswhoseoutputatanytimeisdetermined from the present combination of inputs. The logic gate is the most basicbuildingblockofcombinationallogic.ThelogicalfunctionperformedbyacombinationalcircuitisfullydefinedbyasetofBooleanexpressions.

Sequential logic circuit comprises both logic gates and the state of storageelementssuchasflip-flops.Asaconsequence,theoutputofasequentialcircuitdependsnotonlyonpresentvalueofinputs butalsoonthepaststateofinputs.

Inthepreviouschapter,wehavediscussedbinarynumbers,codes,BooleanalgebraandsimplificationofBooleanfunctionandlogicgates.Inthischapter,formulationandanalysisofvarioussystematicdesignsofcombinationalcircuitswillbediscussed.

Acombinationalcircuitconsistsofinputvariables,logicgates,andoutputvariables. The logic gates accept signals from inputs and output signals are generatedaccording to the logic circuits employed in it. Binary information from the given datatransforms to desired output data in this process. Both input and output are obviouslythe binary signals, *i.e.*, both the input and output signals are of two possible states, logic1andlogic0.



Blockdiagramofacombinationallogiccircuit

For$n$numberofinputvariablestoacombinationalcircuit,2^npossiblecombinat ionsofbinaryinputstatesarepossible.Foreachpossiblecombination,thereisone and only one possible output combination. A combinational logic circuit can bedescribed by *m* Boolean functions and each output can be expressed in terms of *n* inputvariables.

DESIGNPROCEDURE:

Anycombinationalcircuitcanbedesignedbythefollowingstepsofdesignprocedure.

1. Theproblemisstated.
2. Identifytheinputandoutputvariables.
3. Theinputandoutputvariablesareassignedlettersymbols.
4. Constructionofatruthtabletomeetinput-outputrequirements.
5. WritingBooleanexpressionsforvariousoutputvariablesintermsofinp utvariables.
6. ThesimplifiedBooleanexpressionisobtainedbyanymethodofminimization —algebraic method,Karnaughmapmethod,ortabulationmethod.
7. Alogicdiagramisrealizedfromthesimplifiedbooleanexpressionusinglog icgates.

Thefollowingguidelinesshouldbefollowedwhilechoosingthepreferredformforhard wareimplementation:

1. Theimplementationshouldhavetheminimumnumberofgates,withthegate susedhavingtheminimumnumberofinputs.
2. Thereshouldbeaminimumnumberofinterconnections.
3. Limitationonthedrivingcapabilityofthegatesshouldnotbeignored.

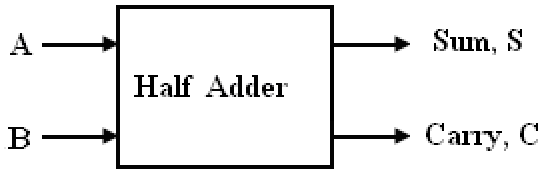
ARITHMETICCIRCUITS–BASICBUILDINGBLOCKS:

Inthissection,wewilldiscussthosecombinationallogicbuildingblocksthatca nbe used to perform addition and subtraction operations on binary numbers. Additionand subtraction are the two most commonly used arithmetic operations, as the othertwo, namely multiplication and division, are respectively the processes of repeatedadditionandrepeated subtraction.

Thebasicbuildingblocksthatformthebasisofallhardwareusedtoperformthea rithmeticoperationsonbinarynumbersarehalf-adder,fulladder,half-subtractor,full-subtractor.

Half-Adder:

A half-adder is a combinational circuit that can be used to add two binary bits. Ithas two inputs that represent the two bits to be added and two outputs, with oneproducingtheSUMoutputandtheotherproducingtheCARRY.

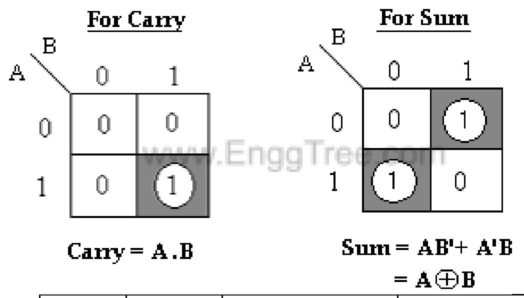


Blockschematicofhalf-adder

Thetruthtableofahalf-adder,showingallpossibleinputcombinationsandthecorrespondingoutputs areshownbelow.

Inputs		Outputs	
A	**B**	**Carry(C)**	**Sum(S)**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truthtableofhalf-adder



K-mapsimplificationforcarryandsum:

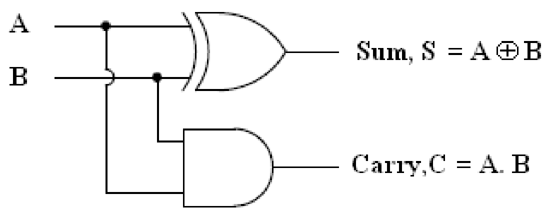
TheBooleanexpressionsfortheSUMandCARRYoutputsaregivenbytheequations,

Sum,S =A’B+AB’=A⊕B

Carry,C=A.B

ThefirstonerepresentingtheSUMoutputisthatofanEX-ORgate,thesecondonerepresentingtheCARRYoutputisthatofanANDgate.

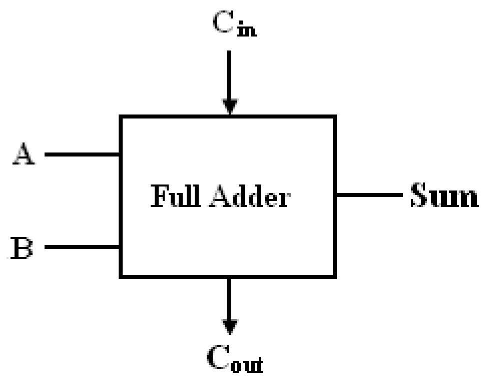
The logicdiagram ofthehalf adderis,



Full- Adder:

Afulladderisacombinationalcircuitthatformsthearithmeticsumofthreeinput bits.Itconsistsof3inputsand2outputs.

Two of the input variables, represent the significant bits to be added. The thirdinputrepresentsthecarryfrompreviouslowersignificantposition.Theblockdiagramoffulladderisgivenby,



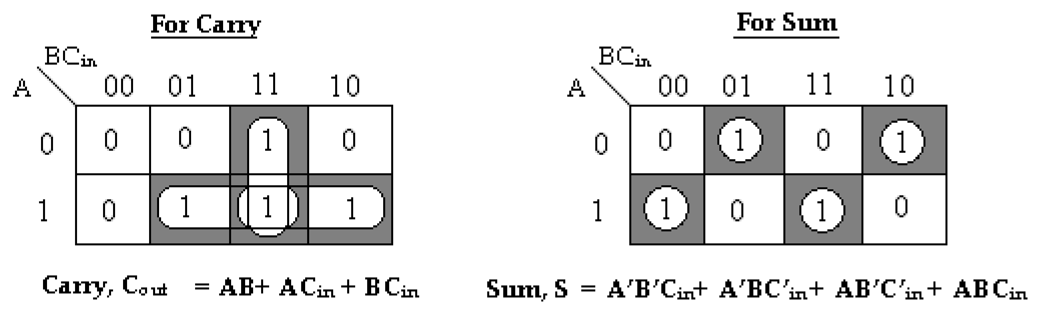
Blockschematicoffull-adder

The full adder circuit overcomes the limitation of the half-adder, which can beused to add two bits only. As there are three input variables, eight different inputcombinationsarepossible.Thetruthtableis shownbelow,

TruthTable:

Inputs			**Outputs**	
A	**B**	**Cin**	**Sum(S)**	**Carry(Cout)**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

ToderivethesimplifiedBooleanexpressionfromthetruthtable,theKarnaughmapmethodisadoptedas,

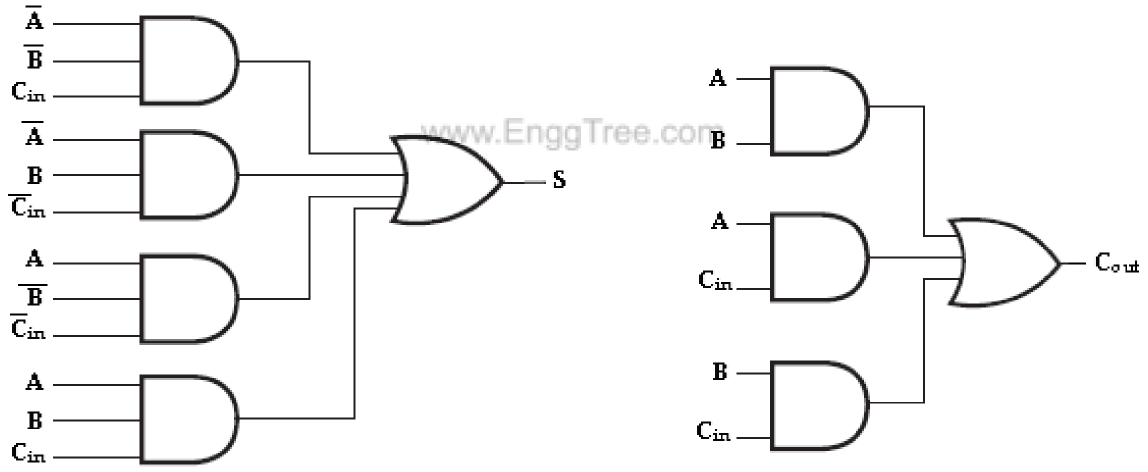

For Carry
For Sum
Carry, C_{out} = AB+ AC_{in} + BC_{in}
Sum, S = A'B'C_{in}+ A'BC'$_{in}$+ AB'C'$_{in}$+ ABC_{in}

TheBooleanexpressionsfortheSUMandCARRYoutputsaregivenbytheequations,

Sum,S =A'B'Cin+A'BC'in+AB'C'in+ABCin

Carry,Cout =AB+ACin+BCin.

Thelogicdiagramfortheabovefunctionsisshownas,



Implementationoffull-adderinSumofProducts

Thelogicdiagram ofthefulladdercanalsobeimplementedwithtwohalf-addersand one ORgate.The S output fromthe secondhalfadderistheexclusive-OR ofCinandtheoutputofthefirsthalf-adder,giving

Sum=Cin⊕(A⊕B) [x⊕y=x'y+xy']

=Cin⊕(A'B+AB')

=C'in(A'B+AB')+Cin(A'B+AB')' [(x'y+xy')'=(xy+x'y')]

=C'in(A'B+AB')+Cin(AB+A'B')

=A‘BC‘in+AB‘C‘in+ABCin+A‘B‘Cin.

andthecarryoutputis,

Carry,Cout=AB+Cin(A’B+AB’)

=AB+A‘BCin+AB‘Cin

=AB(Cin+1)+A‘BCin+AB‘Cin [Cin+1=1]

=ABCin+AB+A‘BCin+AB‘Cin

=AB+ACin(B+B‘)+A‘BCin

=AB+ACin+A‘BCin

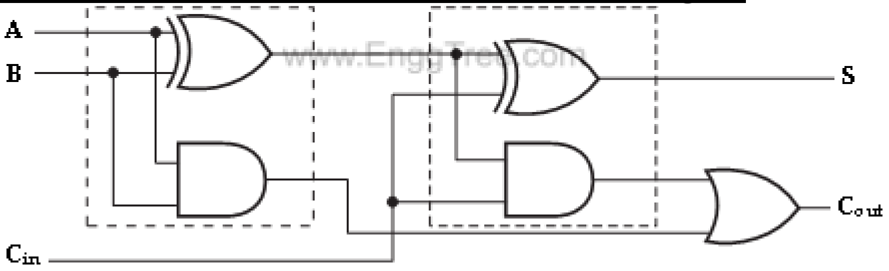
=AB(Cin+1)+ACin+A‘BCin [Cin+1=1]

=ABCin+AB+ACin+A‘BCin

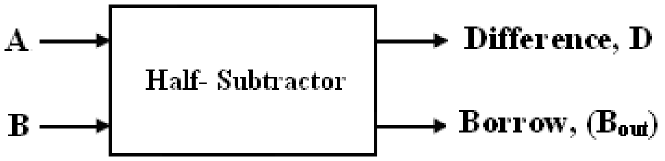
=AB+ACin+BCin(A+A‘)

=AB+ACin+BCin.

Implementationoffulladderwithtwohalf-addersandanORgate



Half-Subtractor:Blockschematicofhalf-subtractor

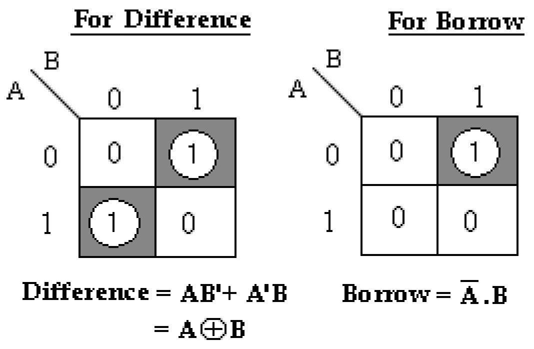


A*half-subtractor*isacombinationalcircuitthatcanbeusedtosubtractonebinarydigitfromanothertoproduceaDIFFERENCEoutputandaBORROWoutput.TheBORROWoutputherespecifieswhethera‗1‘hasbeenborrowedtoperformthesubtraction.Thetruthtableofhalf-subtractor,showingallpossibleinputcombinationsandthecorrespondingoutputsareshownbelow.

Input		Output	
A	B	Difference(D)	Borrow(Bout)
0	0	0	0
0	1	1	1
1	0	1	0

1	1	0	0

K-mapsimplificationforhalfsubtractor:



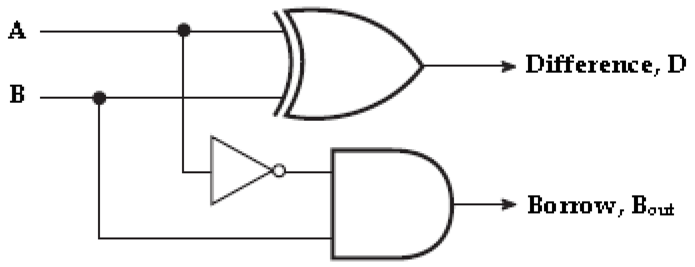
The Boolean expressions for the DIFFERENCE and BORROW outputs are givenbytheequations,

Difference,D= A'B+ AB'=A⊕B

Borrow,Bout =A'.B

The first one representing the DIFFERENCE (**D**)output is that of an exclusive-ORgate, the expression for the BORROW output (**Bout**) is that of an AND gate with input Acomplementedbeforeitisfedtothegate.

The logicdiagramofthehalf adderis,



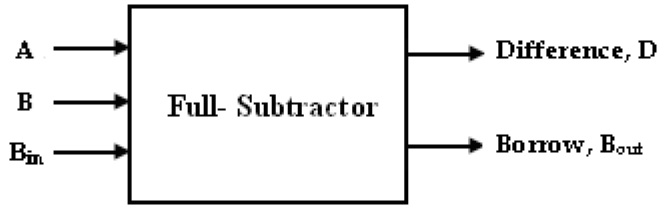
LogicImplementationofHalf-Subtractor

Comparing a half-subtractor with a half-adder, we find that the expressions forthe SUM and DIFFERENCE outputs are just the same. The expression for BORROW inthe case of the half-subtractor is also similar to what we have for CARRY in the case ofthe half-adder. If the input A, ie., the minuend is complemented, an AND gate can beusedtoimplementtheBORROWoutput.

FullSubtractor:

A*fullsubtractor*performssubtractionoperationontwobits,aminuendandasubtrahend, and alsotakes intoconsideration whethera ‗1‘ has alreadybeen borrowedbythepreviousadjacentlowerminuendbitornot.

As a result, there are three bits to be handled at the input of a full subtractor,namely the two bits to be subtracted and a borrow bit designated as Bin. There are twooutputs,namelytheDIFFERENCEoutputDandthe BORROWoutput Bo. TheBORROW output bit tells whether the minuend bit



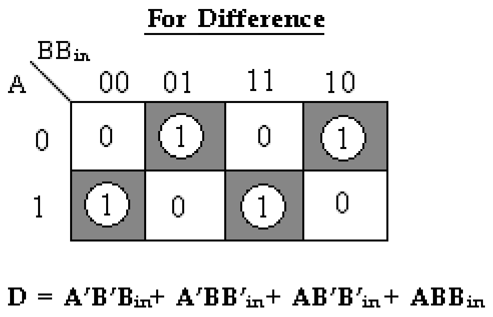
needs to borrow a ‗1‘ from the nextpossiblehigherminuendbit.

Block schematicof full-adder

Thetruthtableforfull-subtractoris,

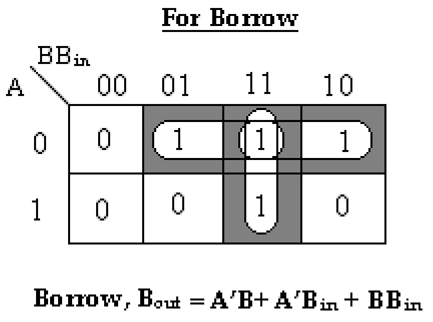
Inputs			Outputs	
A	**B**	**Bin**	**Difference(D)**	**Borrow(Bout)**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-map simplification for full-subtractor:



A \ BB_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Difference, D = A′B′B$_{in}$+ A′BB′$_{in}$+ AB′B′$_{in}$+ ABB$_{in}$



A \ BB_{in}	00	01	11	10
0	0	1	1	1
1	0	0	1	0

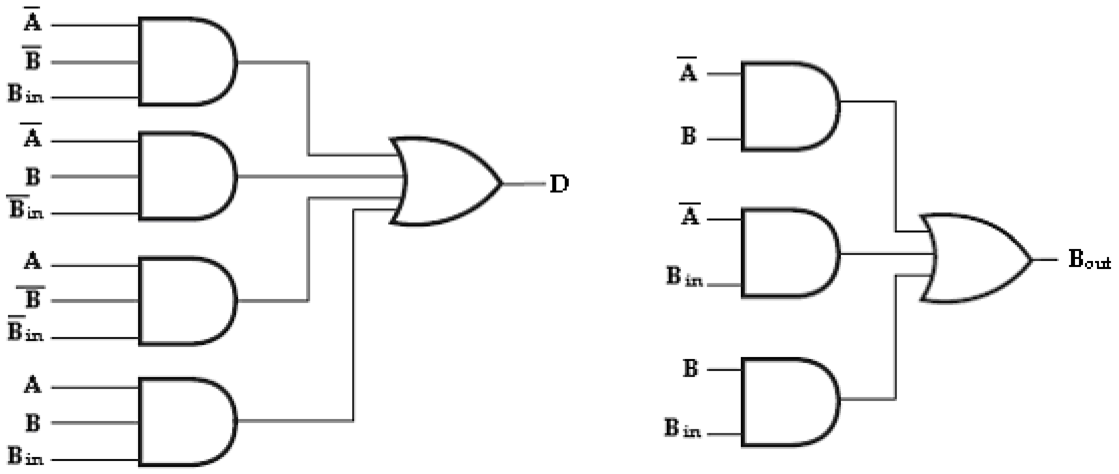
Borrow, B$_{out}$ = A′B+ A′B$_{in}$ + BB$_{in}$

TheBooleanexpressionsfortheDIFFERENCEandBORROWoutputsaregivenbytheequations,

Difference,D =A’B’Bin+A’BB’in+AB’B’in+ABBin

Borrow,Bout =A’B+A’Cin+BBin.

Thelogicdiagramfortheabovefunctionsisshownas,



<u>Implementationoffull-adderinSumofProducts</u>

The logic diagram of the full-subtractor can also be implemented with two half-subtractorsandoneORgate.Thedifference,Doutputfromthesecondhalfsubtractoristheexclusive-ORofBinandtheoutputofthefirsthalf-subtractor, giving

Difference,D=Bin⊕(A⊕B) [x⊕y=x‘y+xy‘]

=**B**in(A‘B+AB‘)

=B‘in(A‘B+AB‘)+Bin(A‘B+AB‘)‘ [(x‘y+xy‘)‘=(xy+x‘y‘)]

=B‘in(A‘B+AB‘)+Bin(AB+A‘B‘)

= A‘BB‘in+ AB‘B‘in+ ABBin+A‘B‘Bin.

andtheborrowoutputis,

Borrow,Bout=A’B+Bin(A’B+AB’)’[(x‘y+xy‘)‘=(xy+x‘y‘)]

=A‘B+Bin(AB+A‘B‘)

= A‘B+ABBin+A‘B‘Bin

=A‘B(Bin+1)+ABBin+A‘B‘Bin [Cin+1=1]

=A‘BBin+A‘B+ABBin+A‘B‘Bin

= A‘B+ BBin(A+A‘)+A‘B‘Bin [A+A‘=1]

=A‘B+BBin+A‘B‘Bin

=A‘B(Bin+1)+BBin+A‘B‘Bin [Cin+1=1]

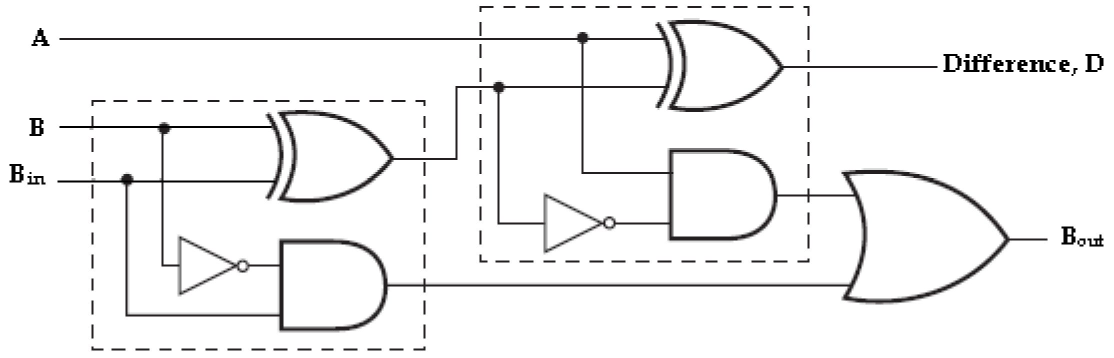
=A‘BBin+A‘B+BBin+A‘B‘Bin

=A‘B+BBin+A‘Bin(B+B‘)

=A‘B+BBin+A‘Bin.

Therefore,

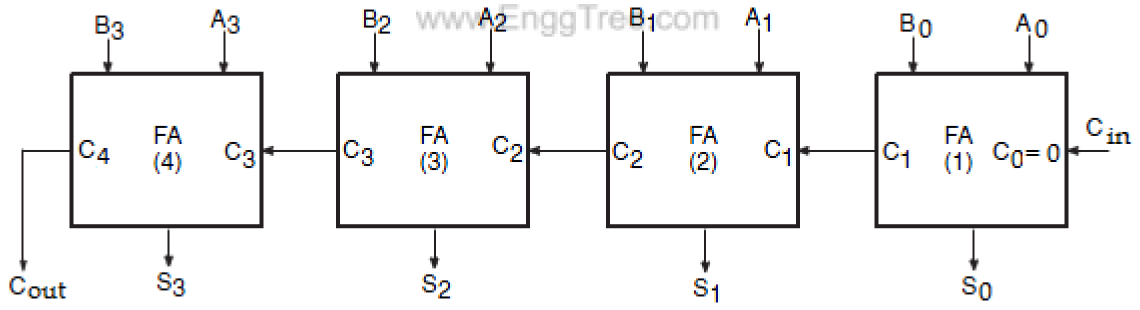
wecanimplementfull-subtractorusingtwohalf-subtractorsandORgateas,



Implementationoffull-subtractorwithtwohalf-subtractorsandanORgate

Four –bit BinaryAdder(ParallelAdder):

The4-bitbinaryadderusingfulladdercircuitsiscapableofaddingtwo4-bitnumbersresulting ina4-bit sumanda carry output as showninfigure below.



4-bitbinaryparallelAdder

Sinceallthebitsofaugendandaddendarefedintotheaddercircuitssimultaneouslyandtheadditionsineachpositionaretakingplaceatthesametime,thiscircuitisknownasparalleladder.

Letthe4-bitwordstobeaddedberepresentedby,A3A2A1A0=1111andB3B2B1B0=0011.

```
Significant place      4 3 2 1
Input carry            1 1 1 0
Augend word A  :       1 1 1 1
Addend word B  :       0 0 1 1
                     ---------
                     1 0 0 1 0 ← Sum
                     ---------
                     ↑
                Output Carry
```

Thebitsareaddedwithfulladders,startingfromtheleastsignificantposition,to formthesumitandcarrybit.TheinputcarryC0intheleastsignificantpositionmustbe 0. The carry output of the lower order stage is connected to the carry input of the nexthigherorderstage.Hencethistypeofadderiscalledripple-carryadder.

In the least significant stage, A0, B0 and C0 (which is 0) are added resulting insumS0andcarryC1.ThiscarryC1becomesthecarryinputtothesecond stage.Similarly in the second stage, A1, B1 and C1 are added resulting in sum S1 and carry C2,in the third stage, A2, B2 and C2 are added resulting in sum S2 and carry C3, in the thirdstage, A3, B3 and C3 are added resulting in sum S3 and C4, which is the output carry.Thusthe circuitresultsin a sum(S3S2S1S0) and acarry output(Cout).

Though the parallel binary adder is said to generate its output immediately afterthe inputs are applied, its speed of operation is limited by the carry propagation delaythrough all stages. However,there are several methods to reduce thisdelay.

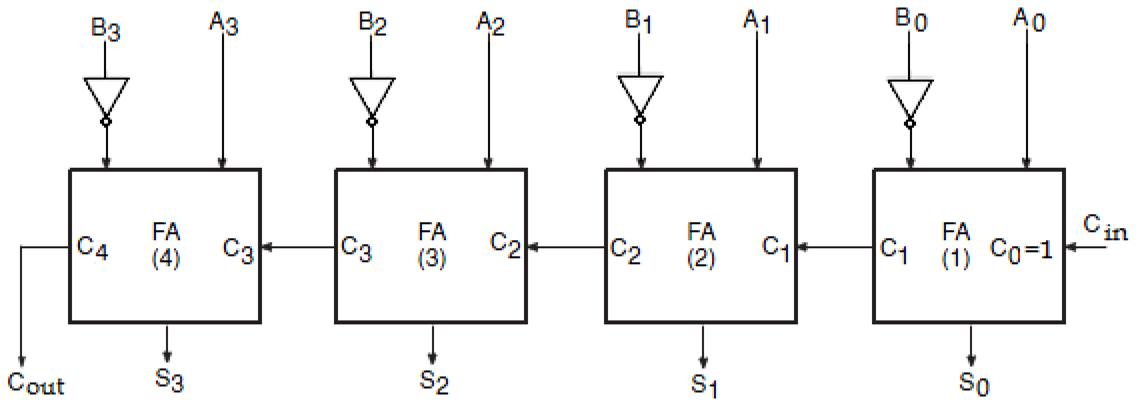
One of the methods of speeding up this process is look-ahead carry additionwhicheliminatestheripple-carrydelay.

BinarySubtractor(ParallelSubtractor):

The subtraction of unsigned binary numbers can be done most conveniently bymeans of complements. The subtraction A-B can be done by taking the 2‘s complementofBandaddingittoA.The2‘scomplementcanbeobtainedbytakingthe1 ‘scomplementandadding1totheleastsignificantpairofbits.

The1‘scomplementcanbeimplementedwithinvertersanda1canbeaddedtoth esumthroughtheinputcarry.

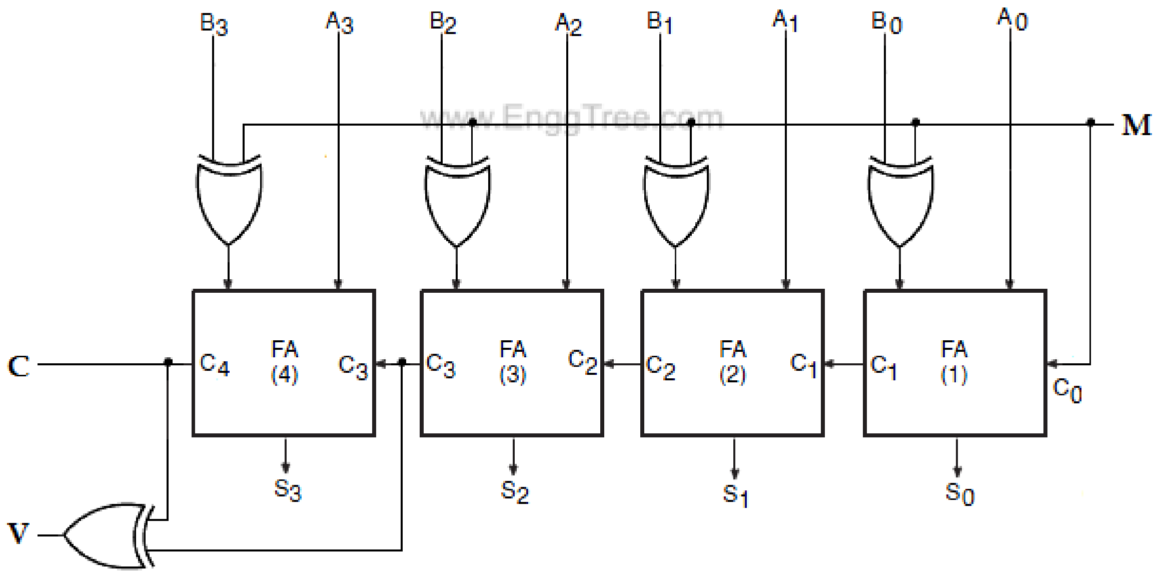
ThecircuitforsubtractingA-Bconsistsofanadderwithinvertersplacedbetween each data input B and the corresponding input of the full adder. The inputcarryC0mustbeequalto1whenperformingsubtraction.Theoperationthusperf ormed becomes A, plus the 1‘s complement of B, plus1. This is equal to A plus the2‘scomplementofB.



4-bitParallelSubtractor

4-Bit ParallelAdder/Subtractor:

The addition and subtraction operation can be combined into one circuit withone common binary adder. This is done by including an exclusive-OR



gate with eachfulladder.A4-bitadderSubtractorcircuitisshownbelow.

ThemodeinputMcontrolstheoperation.WhenM=0,thecircuitisanadderand whenM=1,thecircuitbecomesaSubtractor.Eachexclusive-ORgatereceivesinputMand one of the inputs of B. When M=0, we have B 0= B. The full adders receive thevalueofB,theinputcarryis0,andthecircuitperformsAplusB.WhenM=1,wehave B 1= B‘ and C0=1. The B inputs are all complemented and a 1 is added through theinput carry. The circuit performs the operation A plus the 2‘s complement of B. Theexclusive-ORwithoutputVisfordetectinganoverflow.

DecimalAdder(BCDAdder):

The digital system handles the decimal number in the form of binary codeddecimalnumbers(BCD).ABCDadderisacircuitthataddstwoBCDbitsandproducesasumdigitalsoinBCD.

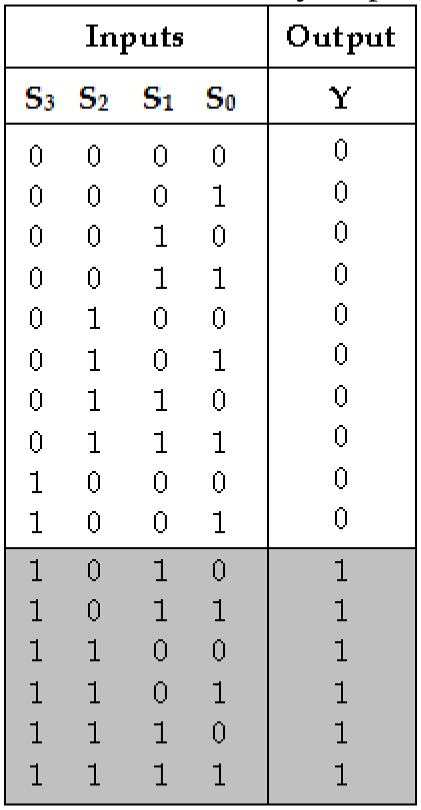
Considerthearithmetic additionof twodecimal digits in BCD,togetherwith aninputcarryfromaprevious stage.Sinceeach inputdigitdoes not exceed9,the outputsumcannotbegreaterthan9+9+1 =19; the1isthesumbeinganinputcarry. Theadderwillformthesuminbinaryandproducearesultthatrangesfrom0through19.

These binary numbers are labeled by symbols K, Z8, Z4, Z2, Z1, K is the carry. Thecolumnsunderthebinarysumlistthebinaryvaluesthatappearintheoutputsofthe4-bitbinaryadder.TheoutputsumofthetwodecimaldigitsmustberepresentedinBCD.

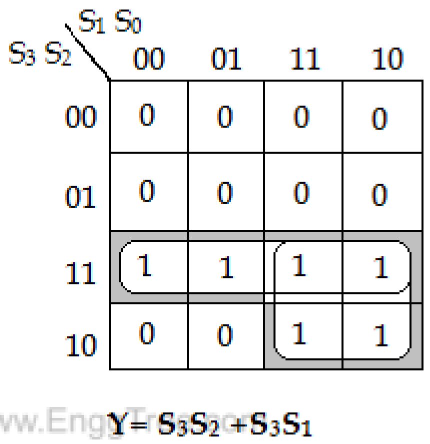
BinarySum					**BCD Sum**					**Decimal**
K	**Z8**	**Z4**	**Z2**	**Z1**	**C**	**S8**	**S4**	**S2**	**S1**	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

In examining the contents of the table, it is apparent that when the binary sum isequal to or less than 1001, the corresponding BCD number is identical, and therefore noconversionis needed. When the binarysum is greaterthan 9(1001), weobtain anon-validBCDrepresentation.Theadditionofbinary6(0110)tothebinarysumconvertsittoth ecorrectBCDrepresentationandalsoproducesanoutputcarryasrequired.

Thelogic circuittodetectsumgreaterthan 9canbedeterminedbysimplifyingthebooleanexpressionofthegiventruthtable.



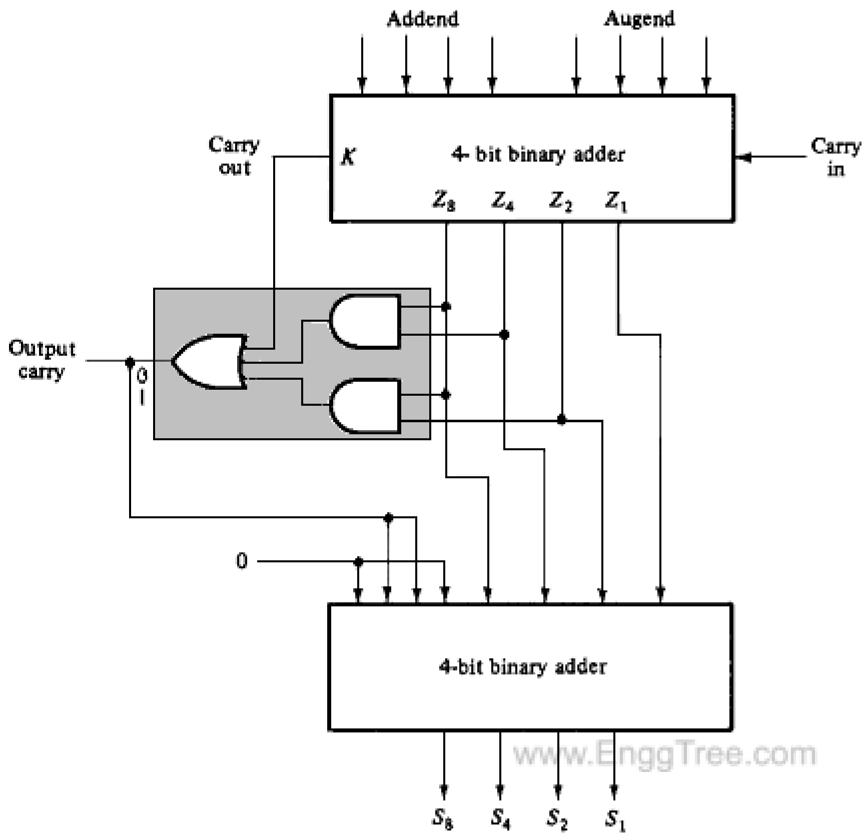
Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



ToimplementBCDadder werequire:

- 4-bitbinaryadderforinitialaddition
- Logiccircuittodetectsumgreaterthan9and
- Onemore4-bitaddertoadd0110$_2$inthesumifthesumisgreaterthan9orcarryis1.

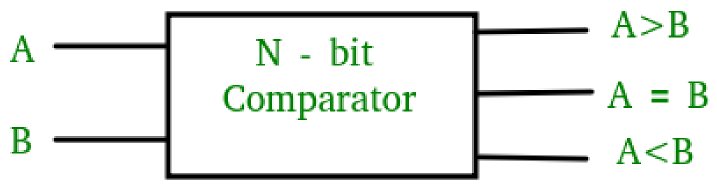
The twodecimal digits, togetherwith the inputcarry, are firstadded in the top4-bit binary adder to provide the binary sum. When the output carry is equal to zero,nothing is added to the binarysum. When itis equal to one, binary0110 is added tothe binary sum through the bottom 4-bit adder. The output carry generated from thebottom adder can be ignored, since it supplies information already available at theoutputcarryterminal.Theoutputcarryfromonestagemustbeconnectedtotheinput carryofthenexthigher-orderstage.



BlockdiagramofBCDadder

MAGNITUDECOMPARATOR:

A ***magnitude comparator*** is a combinational circuit that compares two givennumbers(AandB)anddetermineswhetheroneisequalto,lessthanorgreatertha ntheother.Theoutputisintheformofthreebinaryvariables



Blockdiagramofn-Bit magnitudecomparator

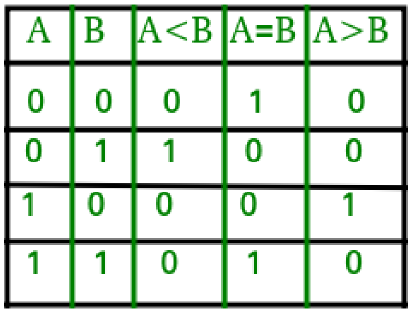
representingtheconditionsA=B,A>BandA<B,ifAandBarethetwonumbers beingcompared.

Forcomparisonoftwo*n*-bitnumbers,theclassicalmethodtoachievetheBoolean expressions requires a truth table of 2^{2n}entries and becomes too lengthy andcumbersome.

1-Bit Magnitude Comparator:

A comparator used to compare two bits is called a single-bit comparator. It consists of two inputs each for two single-bit numbers and three outputs to generate less than, equal to, and greater than between two binary numbers.

The truth table for a 1-bit comparator is given below:



A	B	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

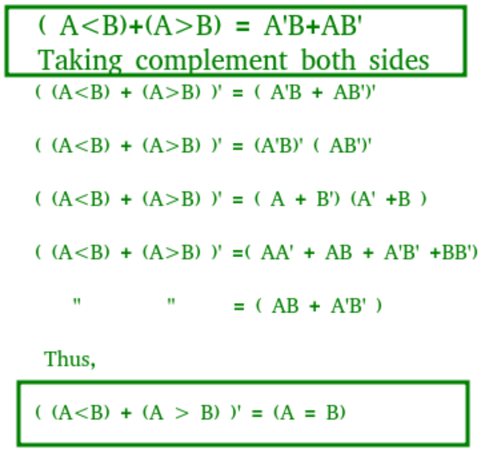
From the above truth table logical expressions for each output can be expressed as follows:

A>B: AB'

A<B: A'B

A=B: A'B' + AB

From the above expressions we can derive the following formula:



$\boxed{(A<B)+(A>B) = A'B+AB' \quad \text{Taking complement both sides}}$

$((A<B) + (A>B))' = (A'B + AB')'$

$((A<B) + (A>B))' = (A'B)'(AB')'$

$((A<B) + (A>B))' = (A + B')(A' + B)$

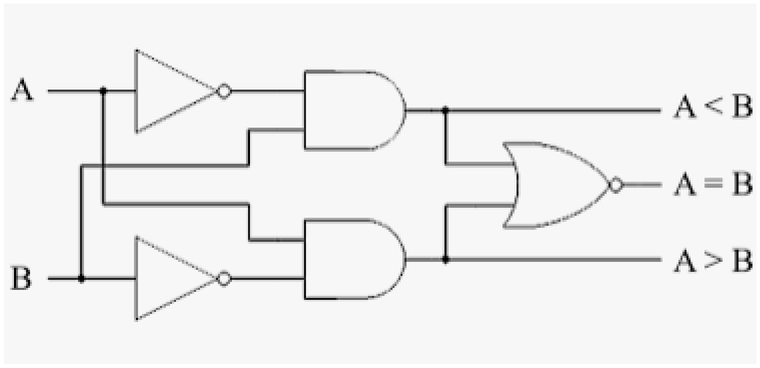
$((A<B) + (A>B))' = (AA' + AB + A'B' + BB')$

" " $= (AB + A'B')$

Thus,

$\boxed{((A<B) + (A > B))' = (A = B)}$

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below:



2-bitMagnitudeComparator:

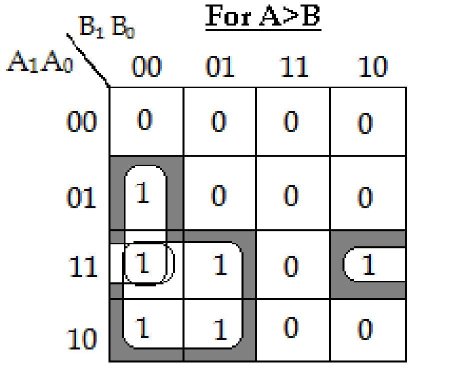
Thetruthtableof2-bitcomparatorisgivenintablebelow—

Truthtable:

Inputs				**Outputs**		
A3	**A2**	**A1**	**A0**	**A>B**	**A=B**	**A<B**
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

K-mapSimplification**:**

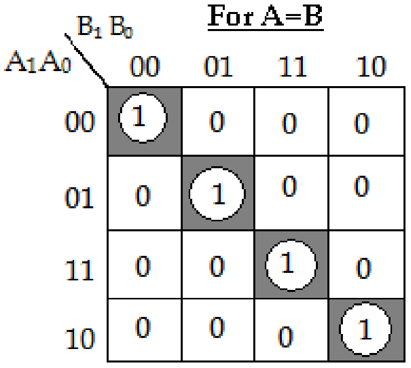
For A>B



A_1A_0 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$A>B = A_0B_1'B_0' + A_1B_1' + A_1A_0B_0'$

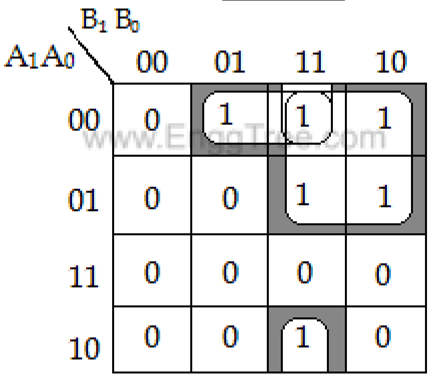
For A=B



A_1A_0 \ B_1B_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$$A=B = A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0B_1B_0 + A_1A_0'B_1B_0'$$
$$= A_1'B_1'(A_0'B_0' + A_0B_0) + A_1B_1(A_0B_0 + A_0'B_0')$$
$$= (A_0 \odot B_0)(A_1 \odot B_1)$$

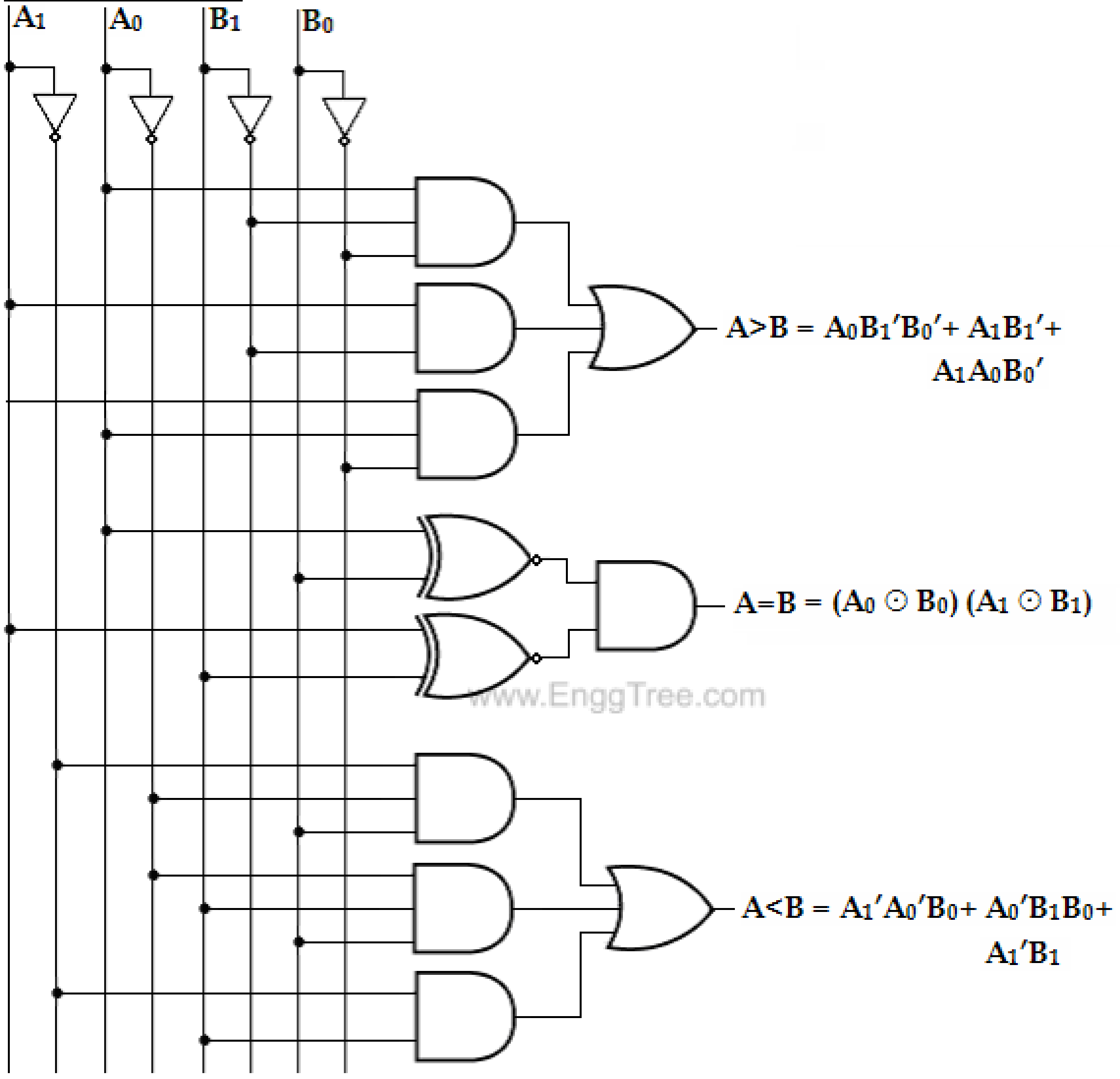
For A<B



A_1A_0 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$A<B = A_1'A_0'B_0 + A_0'B_1B_0 + A_1'B_1$

LogicDiagram:

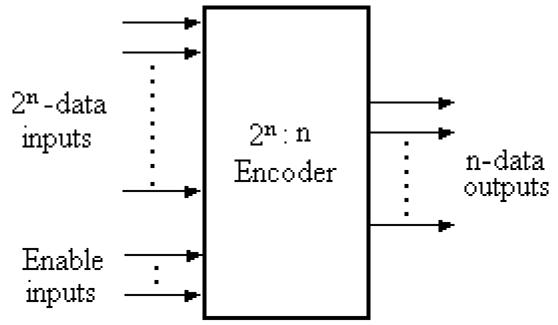


A_1
A_0
B_1
B_0
$A>B = A_0B_1'B_0' + A_1B_1' + A_1A_0B_0'$
$A=B = (A_0 \odot B_0)(A_1 \odot B_1)$
www.EnggTree.com
$A<B = A_1'A_0'B_0 + A_0'B_1B_0 + A_1'B_1$

2- bitMagnitudeComparator

ENCODERS:

An encoder is a digital circuit that performs the inverse operation of a decoder.Hence,theoppositeofthedecodingprocessiscalledencoding.Anencoderis acombinationalcircuitthatconvertsbinaryinformationfrom2^{n}inputlinestoamaximumof‗n‘uniqueoutputlines.
Thegeneralstructureofencodercircuitis–



<u>GeneralstructureofEncoder</u>

It has 2^{n} input lines, only one which 1 is active at any time and ‗n‘ output lines. Itencodes one of the active inputs to a coded binary output with ‗n‘ bits. In an encoder,thenumberofoutputsislessthanthenumberofinputs.

<u>Octal-to-BinaryEncoder:</u>

It has eight inputs (one for each of the octal digits) and the three outputs thatgenerate the corresponding binary number. It is assumed that only one input has avalueof1atanygiventime.

Inputs								Outputs		
D0	D1	D2	D3	D4	D5	D6	D7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

The encoder can be implemented with OR gates whose inputs are determineddirectly from the truth table. Output z is equal to 1, when the input octal digit is 1 or 3or5or7.Outputyis 1foroctaldigits2,3,6,or7andtheoutputis1fordigits4,5,6or 7.Theseconditionscanbeexpressedbythefollowingoutput Booleanfunctions:

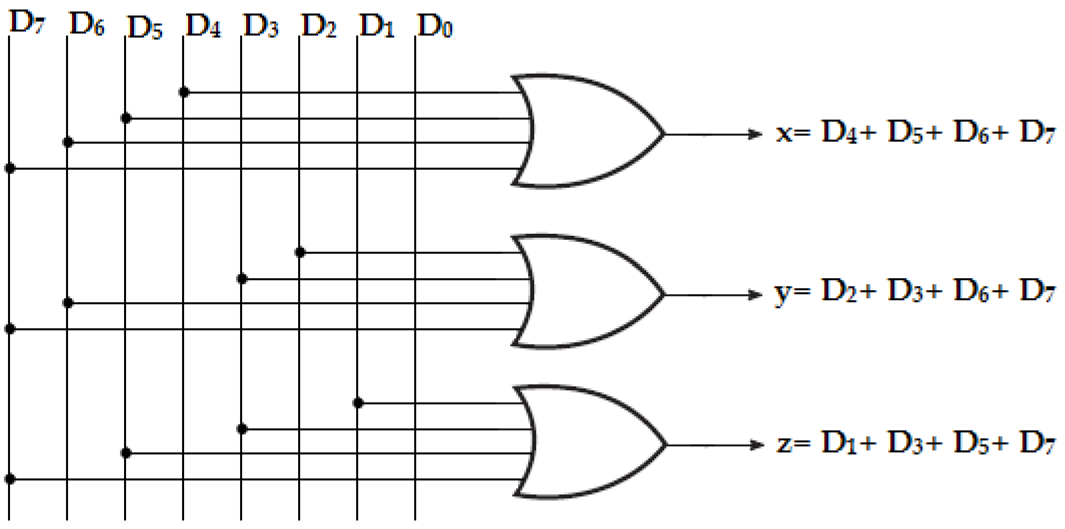
z=D1+D3+D5+D7

y=D2+D3+D6+D7

x=D4+D5+D6+D7

The encoder can be implemented with three OR gates. The encoder defined inthe below table, has the limitation that only one input can be active at any given time. Iftwoinputsareactivesimultaneously,theoutputproducesanundefinedcombinatio n.

For eg., if D3 and D6 are 1 simultaneously, the output of the encoder may be 111.This does not represent either D6 or D3. To resolve this problem, encoder circuits mustestablish an input priority to ensure that only one input is encoded. If we establish ahigher priority for inputs with higher subscript numbers and if D3and D6are 1 at thesame time, the output will be 110 becauseD6hashigher prioritythan D3.



<u>Octal-to-BinaryEncoder</u>

Another problem in the octal-to-binary encoder is that an output with all 0's isgenerated when all the inputs are 0; this output is same as when D0is equal to 1. Thediscrepancycan be resolvedbyproviding one moreoutput toindicatethatatleastoneinputisequalto1.

PriorityEncoder:

A priority encoder is an encoder circuit that includes the priority function. Inpriorityencoder,iftwoormoreinputsareequalto1atthesametime,theinputhavingthehighestprioritywilltakeprecedence.

In addition to the two outputs x and y, the circuit has a third output, V (valid bitindicator).Itis set to 1 whenoneormore inputs areequal to 1.If allinputs are0, thereisnovalidinputandVisequalto0.Thehigherthesubscriptnumber,higherthepriority oftheinput.InputD3,hasthe highest priority. So, regardless of the values of the other inputs, whenD3is 1, theoutputforxyis11.

D2hasthenextprioritylevel.Theoutputis10,ifD2=1providedD3=0.TheoutputforD1isgeneratedonlyifhigherpriorityinputsare0,andsoondownthe prioritylevels.

Truthtable:

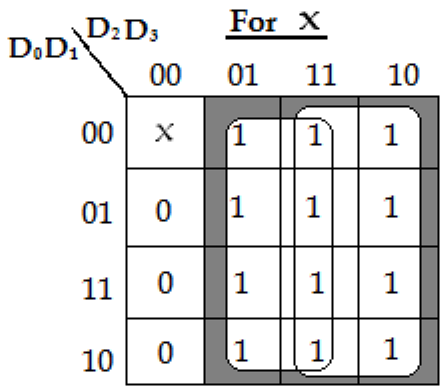
Inputs				**Outputs**		
D0	**D1**	**D2**	**D3**	**x**	**y**	**V**
0	0	0	0	x	x	0
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

Although the above table has only five rows, when each don‘t care condition isreplaced first by 0 and then by 1, we obtain all 16 possible input combinations. Forexample,thethirdrowinthetablewithX100representsminterms0100and1100.Thedon‘tcareconditionis replacedby0and 1asshown in thetable below.

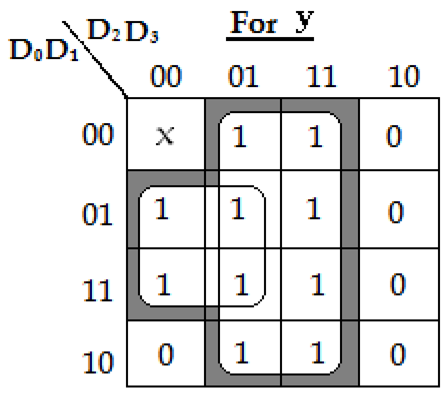
ModifiedTruthtable:

Inputs				Outputs		
D0	D1	D2	D3	x	y	V
0	0	0	0	x	x	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0			
0	0	1	0	1	0	1
0	1	1	0			
1	0	1	0			
1	1	1	0			
0	0	0	1	1	1	1
0	0	1	1			
0	1	0	1			
0	1	1	1			
1	0	0	1			
1	0	1	1			
1	1	0	1			
1	1	1	1			

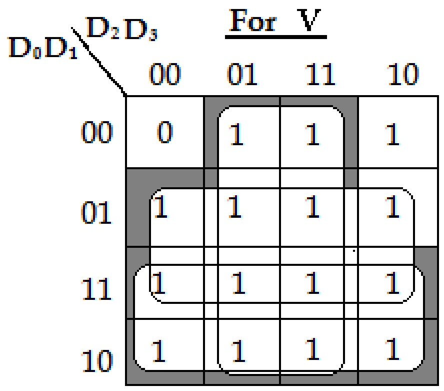
K-mapSimplification:



$x = D_2 + D_3$

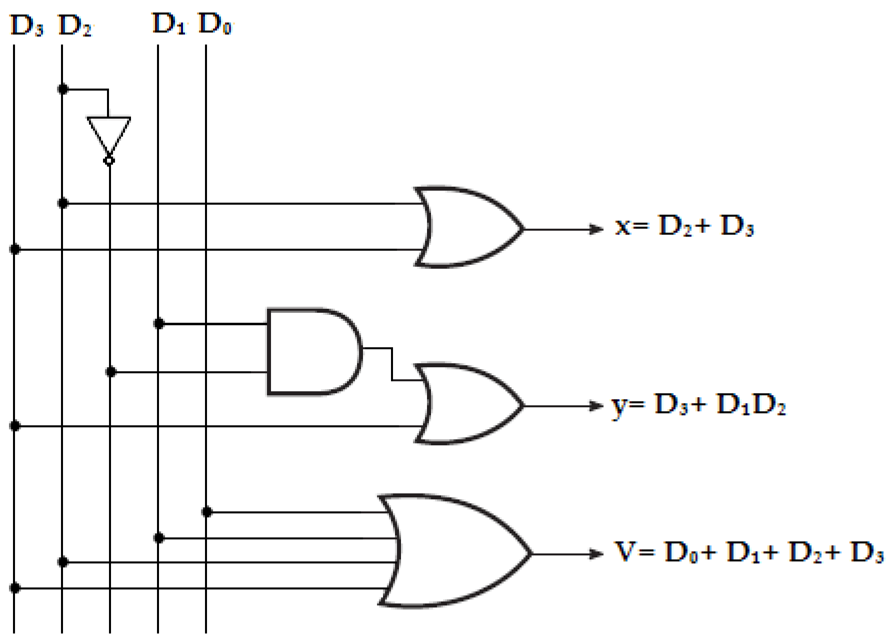


$y = D_3 + D_1 D_2'$



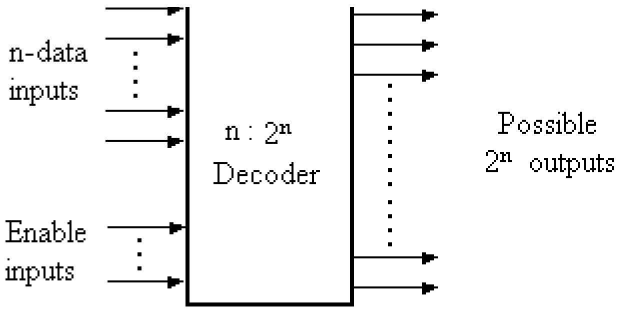
$V = D_0 + D_1 + D_2 + D_3$

ThepriorityencoderisimplementedaccordingtotheaboveBooleanfunctions.



InputPriorityEncoder

DECODERS:



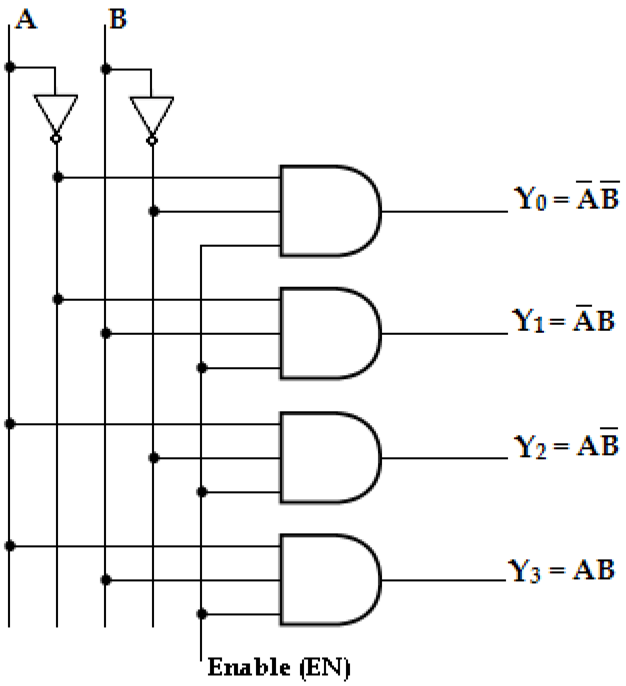
A decoder is a combinational circuit that converts binary information from _n‘inputlinestoamaximumof_2^n‘uniqueoutputlines.Thegeneralstructureofdecodercircuitis–

Generalstructureofdecoder

Theencodedinformationispresentedas_n‘inputsproducing_2^n‘possibleoutputs. The 2^noutput values are from 0 through 2^n-1. A decoder is provided withenable inputs to activate decoded output based on data inputs. When any one enableinputisunasserted,alloutputsofdecoderaredisabled.

BinaryDecoder(2to4decoder):

A binary decoder has _n‘ bit binary input and a one activated output out of 2^noutputs. A binary decoder is used when it is necessary to activate exactly one of 2^noutputsbasedonann-bitinputvalue.



2-to-4Linedecoder

Herethe2inputsaredecodedinto4outputs,eachoutputrepresentingoneoftheminter msofthetwoinputvariables.

Input s			**Output s**			
Enabl e	**A**	**B**	**Y 3**	**Y 2**	**Y 1**	**Y 0**
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

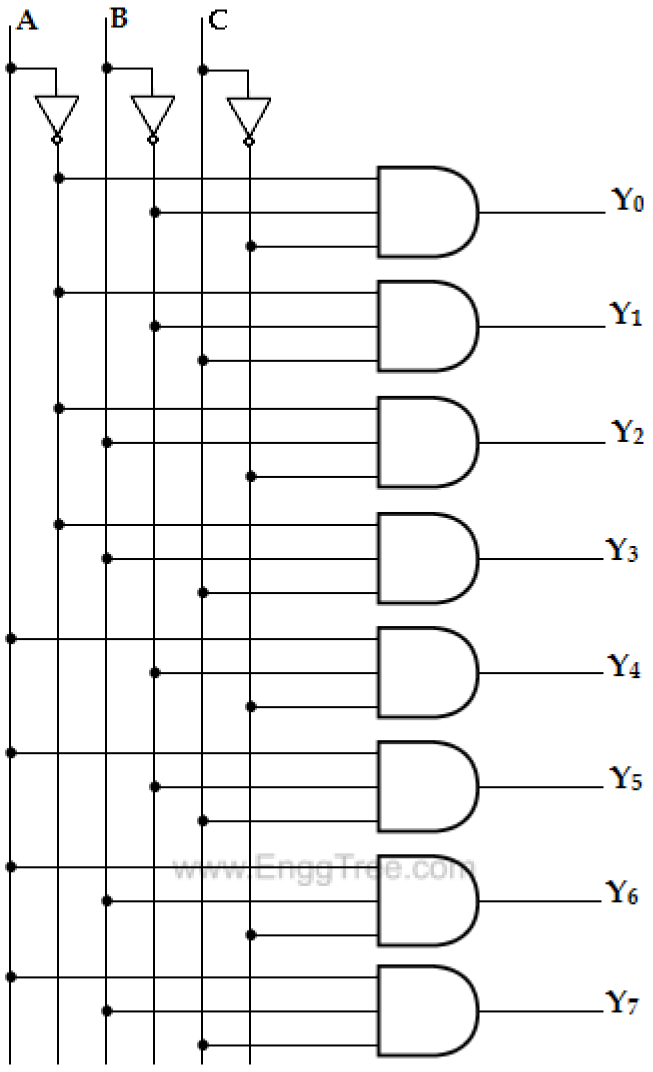
Asshowninthetruthtable,ifenableinputis1(EN=1)onlyoneoftheoutputs(Y0–Y3),isactiveforagiveninput.
TheoutputY0isactive,ie.,Y0=1wheninputs
A=B=0,Y1isactivewheninputs,
A=0andB=1,
Y2isactive,wheninputA=1andB=0,
Y3isactive,
wheninputsA=B=1.

3 to-8LineDecoder:

A3-to-8linedecoderhasthreeinputs(A,B,C)andeightoutputs(Y0-Y7).Basedonthe3inputsoneoftheeightoutputsisselected.

Thethreeinputsaredecodedintoeightoutputs,eachoutputrepresentingoneofthe mintermsofthe3-inputvariables.Thisdecoderisusedforbinary-to-octalconversion.Theinputvariablesmayrepresentabinarynumberandtheoutputswillre present the eight digits in the octal number system. The output variables are mutuallyexclusivebecauseonlyone outputcanbeequalto1 atanyone time.Theoutputlinewhosevalueisequalto1representsthemintermequivalentofthebinar ynumberpresentlyavailableintheinputlines.

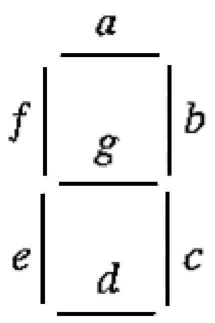
Inputs			**Outputs**							
A	**B**	**C**	**Y0**	**Y1**	**Y2**	**Y3**	**Y4**	**Y5**	**Y6**	**Y7**
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



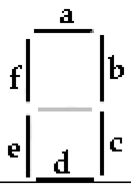
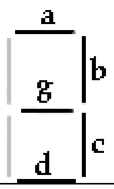
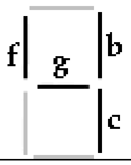
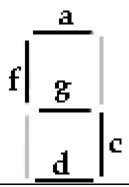
3-to-8linedecoder

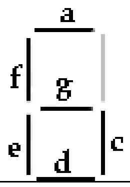
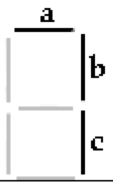
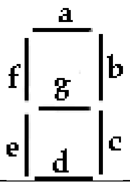
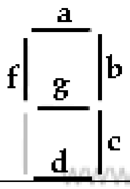
BCDto7-SegmentDisplayDecoder:

A seven-segment display isnormally usedfordisplaying any one of the decimaldigits,0through9.ABCD-to-sevensegmentdecoderacceptsadecimaldigitinBCDandgeneratesthecorrespondingseven-segmentcode.



Eachsegmentismadeupofamaterialthatemitslightwhencurrentispassedthrou ghit.Thesegmentsactivatedduringeachdigit displayaretabulatedas—

Dig it	Displa y	Segments Activated
0		a,b,c, d,e,f
1		b,c
2		a, b, d, e, g
3		a, b,c,d,g
4		b,c,f, g
5		a,c,d,f,g

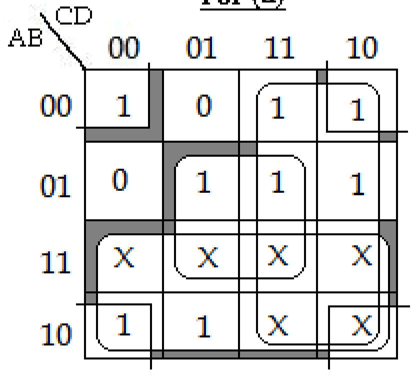
6		a,c,d,e,f,g
7		a,b,c
8		a, b,c,d,e,f, g
9		a,b,c,d,f,g

Truthtable:

	BCDcode				7- Segmentcode						
Digit	**A**	**B**	**C**	**D**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

K-mapSimplification:

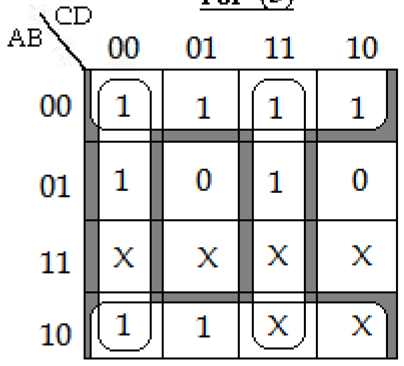
For (a)



AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

a= A+ C+ BD+ B'D'

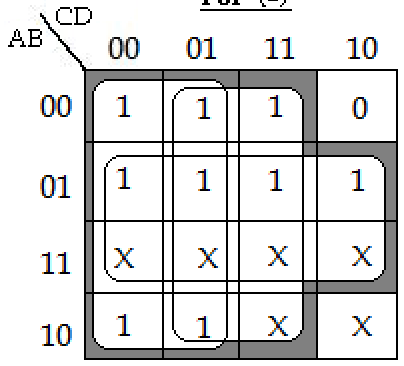
For (b)



AB \ CD	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	X	X	X	X
10	1	1	X	X

b= B'+ C'D'+ CD

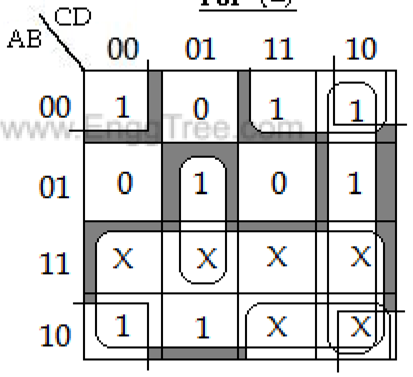
For (c)



AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

c= B+ C'+ D

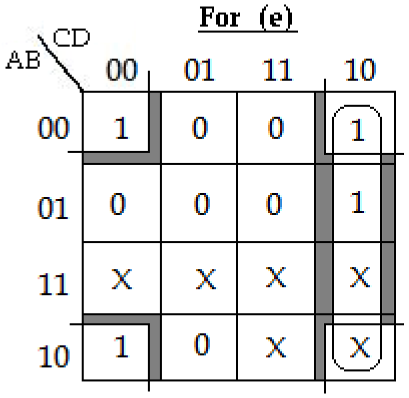
For (d)



AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	X	X	X	X
10	1	1	X	X

d= B'D'+ CD'+ BC'D+ B'C+ A

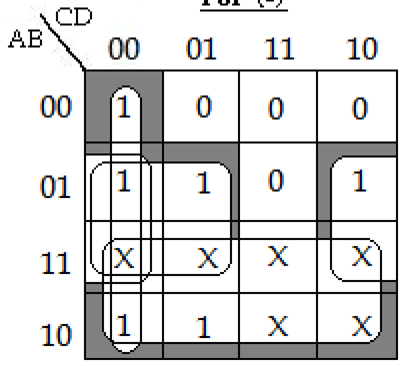
For (e)



AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	X	X	X	X
10	1	0	X	X

e= B'D'+ CD'

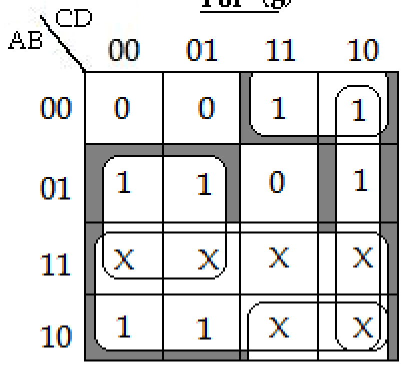
For (f)



AB \ CD	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

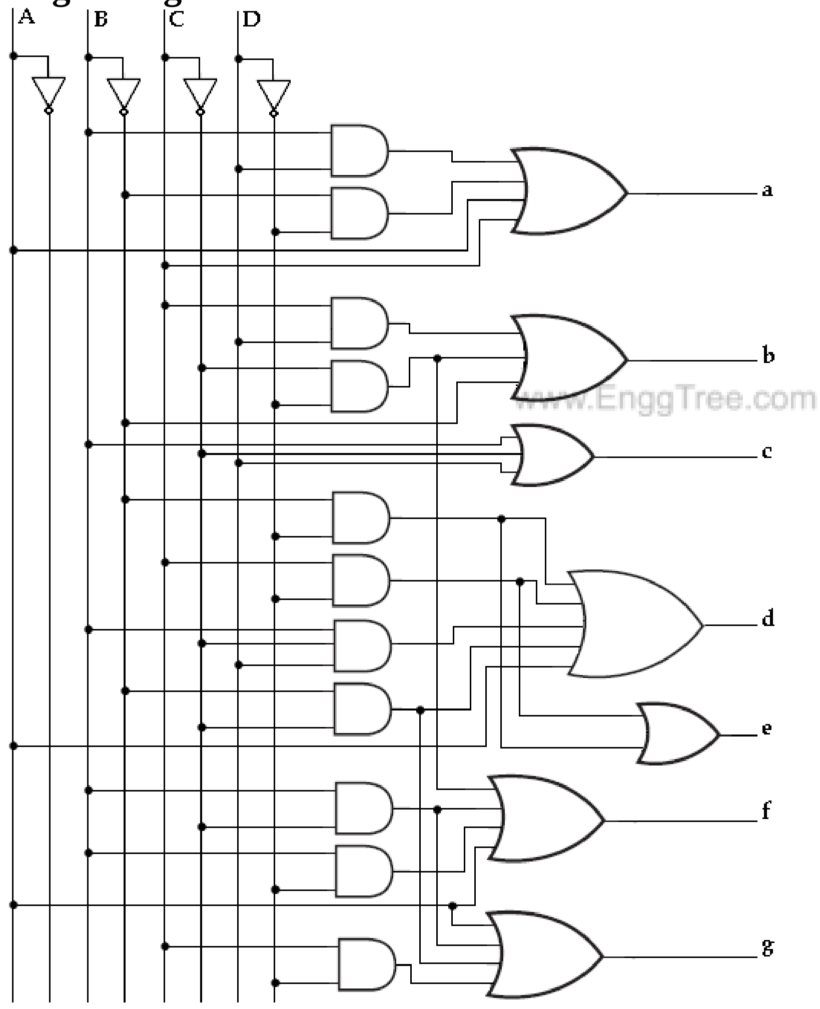
f= A+ C'D'+ BC'+ BD'

For (g)



g= A+ BC'+ B'C+ CD'

Logic Diagram



BCDto7-segmentdisplaydecoder

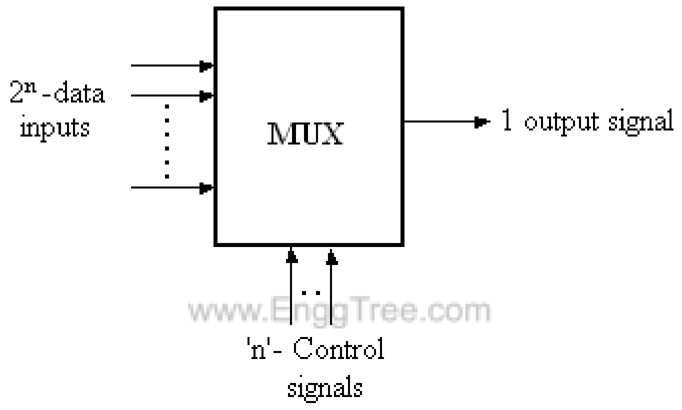
Applicationsofdecoders:

1. Decodersareusedincountersystem.
2. Theyareusedinanalogtodigitalconverter.
3. Decoderoutputscanbeusedtodriveadisplaysystem.

MULTIPLEXER:(DataSelector)

A ***multiplexer*** or *MUX*, is a combinational circuit with more than one input line,oneoutputlineandmorethanoneselectionline.Amultiplexerselectsbinaryinformation present from one of many input lines, depending upon the logic status ofthe selection inputs, and routes it to the output line. Normally, there are 2^n input linesand n selection lines whose bit combinations determine which input is selected. ThemultiplexerisoftenlabeledasMUXinblockdiagrams.

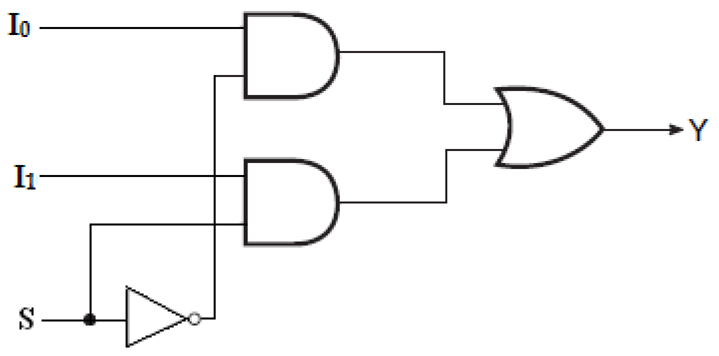
A multiplexer is also called a **data selector**, since it selects one of many inputsandsteersthebinaryinformationto theoutputline.



<u>BlockdiagramofMultiplexer</u>

<u>2-to-1-lineMultiplexer</u>:

Thecircuithastwodatainputlines,oneoutputlineandoneselectionline,S.WhenS= 0,theupperANDgateisenabledandI_0hasapathtotheoutput.
When S=1,thelowerAND gateisenabledandI_1has apathtotheoutput.



Themultiplexeractslikeanelectronicswitchthatselectsoneofthetwosources.

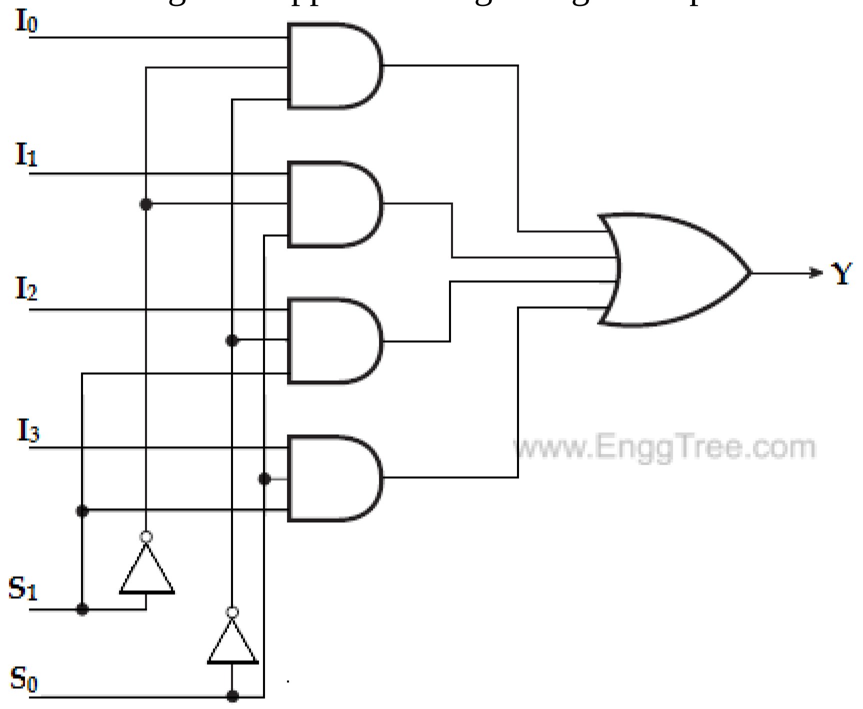
<u>Truthtable:</u>

S	Y
0	I_0
1	I_1

4-to-1-lineMultiplexer:

A 4-to-1-line multiplexer has four (2^n) input lines, two (n) select lines and oneoutput line. It is the multiplexer consisting of four input channels and information ofone of the channels can be selected and transmitted to an output line according to theselect inputs combinations. Selection of one of the four input channel is possible by twoselectioninputs.

EachofthefourinputsI0throughI3,isappliedtooneinputofANDgate.Selection lines S1and S0are decoded to select a particular AND gate. The outputs of theANDgateareappliedtoasingleORgatethatprovidesthe1-lineoutput.



4-to-1-LineMultiplexer

Functiontable:

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

To demonstrate the circuit operation, consider the case when S1S0= 10. The ANDgateassociatedwithinputI2hastwoofitsinputsequalto1andthethirdinputconnecte dtoI2.TheotherthreeANDgateshaveatleastoneinputequalto0,whichmakestheiroutput sequalto0.TheORoutputisnowequaltothevalueofI2,providingapathfromtheselectedin puttotheoutput.

The data output is equal to I0 only if S1= 0 and S0= 0;

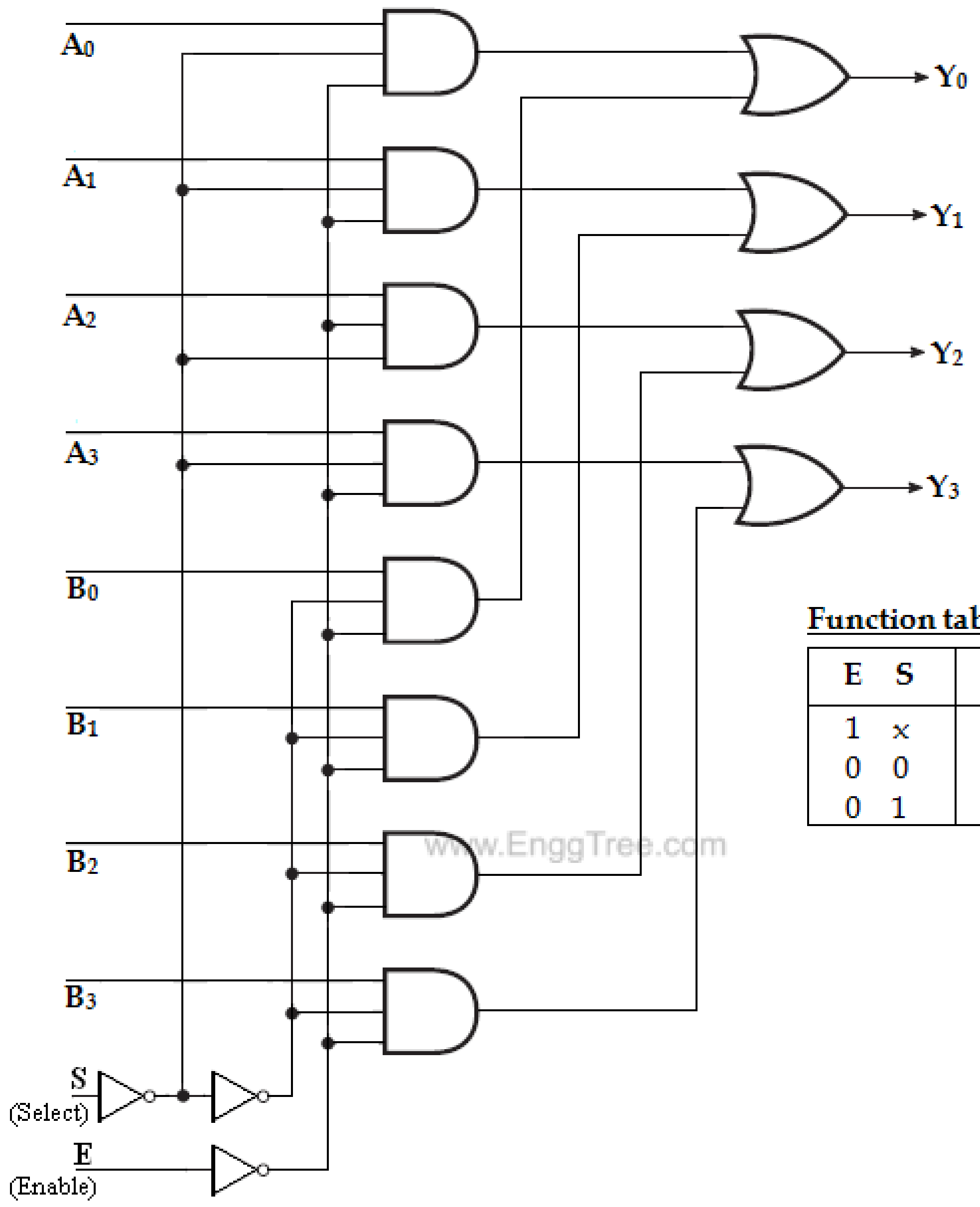
Y= I0S1‘S0‘.The data output is equal to I1 only if S1= 0 and S0= 1; Y= I1S1‘S0.The data output is equal to I2 only if S1= 1 and S0= 0; Y= I2S1S0‘.ThedataoutputisequaltoI3onlyifS1=1andS0=1;Y =I3S1S0.

WhenthesetermsareORed,thetotalexpressionforthedataoutputis,

Y=I0S1’S0’+I1S1’S0+I2S1S0’+I3S1S0.

As in decoder, multiplexers may have an enable input to control the operation ofthe unit. When the enable input is in the inactive state, the outputs are disabled, andwhenit isin the activestate, the circuit functionsasa normalmultiplexer.

<u>Quadruple 2-to-1LineMultiplexer</u>:



Function table:

E	S	Output Y
1	×	All 0's
0	0	Select A
0	1	Select B

This circuit has four multiplexers, each capable of selecting one of two inputlines. Output Y0 can be selected to come from either A0 or B0. Similarly, output Y1 mayhave the value of A1 or B1, and so on. Input selection line, S selects one of the lines ineachofthefourmultiplexers.TheenableinputEmustbeactivefornormaloperation.

Although the circuit contains four 2-to-1-Line multiplexers, it is viewed as acircuit that selects one of two 4-bit sets of data lines. The unit is enabled when E= 0.ThenifS=0,thefourAinputshaveapathtothefouroutputs.Ontheotherhand,ifS=1, the four B inputs are applied to the outputs. The outputs have all 0‘s when E= 1,regardlessofthevalueofS.

Application:

Themultiplexerisaveryusefu lMSIfunctionandhasvariousrangesofapplicati ons in data communication. Signal routing and data communication are theimportantapplicationsofamultiplexer.Itisusedforconnectingtwoormoresource stoguide to a single destination among computer units and it is useful for constructing acommon bus system. One of the general properties of a multiplexer is that Booleanfunctionscanbeimplementedbythisdevice.

ImplementationofBooleanFunctionusingMUX:

AnyBooleanorlogicalexpressioncanbeeasilyimplementedusingamultiplex er. If a Boolean expression has (n+1) variables, then ‗n' of these variables canbe connected to the select lines of the multiplexer. The remaining single variable alongwith constants 1 and 0 is used as the input of the multiplexer. For example, if C is thesinglevariable,thentheinputsofthemultiplexersareC,C',1and0.Bythismethoda nylogicalexpression canbeimplemented.

In general, a Boolean expression of (n+1) variables can be implemented using amultiplexerwith2^ninputs.

1. **Implementthefollowingboolean functionusing4:1multiplexer,F(A,B,C)=∑m(1,3,5,6).**

Solution:

Variables,n=3(A,B,C)Se lectlines=n-1=2(**S1,S0**)

2^{n-1}toMUXi.e.,2^2to1=4to1MUXInputli nes=2^{n-1}=2^2=4(**D0,D1,D2,D3**)

Implementationtable:

ApplyvariablesAandBtotheselectlines.Theproceduresforimplementingthefunct ionare:

i. Listtheinputofthemultiplexer
ii. Listunderthemalltheminterms intworowsasshownbelow.

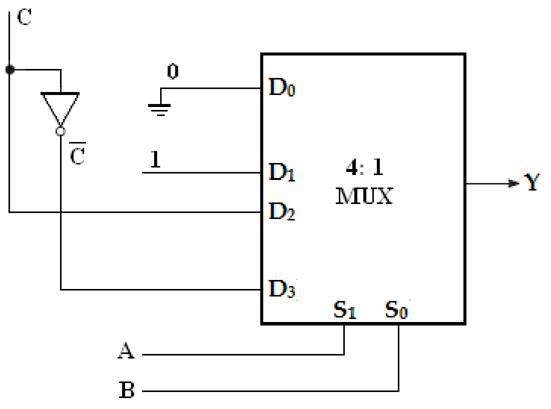
The first half of the minterms is associated with A' and the second half with A. Thegivenfunctionisimplementedbycirclingthemintermsofthefunctionandapplyi ngthefollowingrulestofindthevaluesfortheinputsofthemultiplexer.

1. Ifboththemintermsinthecolumnarenotcircled,apply0tothecorrespondingin

put.

2. Ifboththemintermsinthecolumnarecircled,apply1tothecorrespondinginput.
3. Ifthebottommintermiscircledandthetopisnotcircled,applyCtotheinput.
4. Ifthetopmintermiscircledandthebottomis notcircled,applyC‘totheinput.

MultiplexerImplementation:



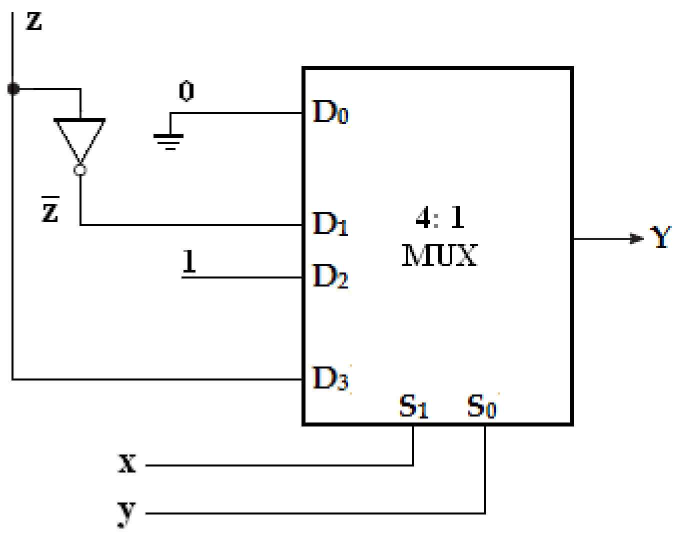
2.F(x,y,z)=∑m(1,2,6,7)

Solution:

Implementationtable:

	D_0	D_1	D_2	D_3
$\bar{z}$	0	1	2	3
z	4	5	6	7
	0	$\bar{z}$	1	z

Multiplexer Implementation:



3.**F(A,B,C) =∑m(1,2,4,5)**

<u>Solution:</u>

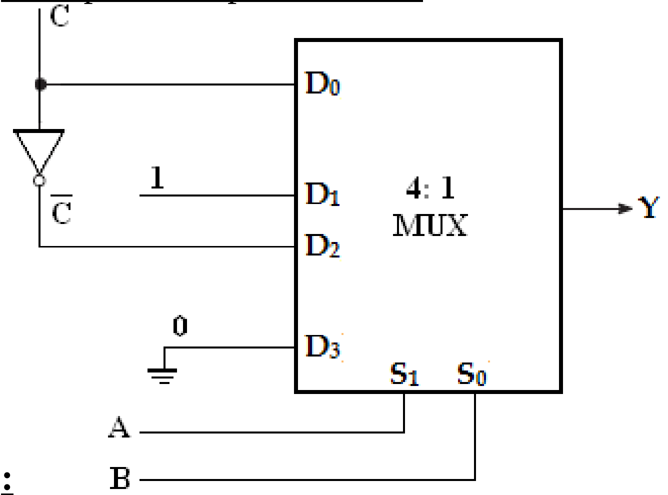
Variables,n=3(A,B,C)Selectli
nes=n-1=2(**S1,S0**)

2^{n-1}toMUXi.e.,2^{2}to1=4to1MUXInputli
nes=2^{n-1}=2^{2}=4(**D0,D1,D2,D3**)

<u>Implementationtable:</u>

	D_0	D_1	D_2	D_3
$\bar{C}$	0	1	2	3
C	4	5	6	7
	C	1	$\bar{C}$	0

<u>Multiplexer Implementation</u>



:

4.F(P,Q,R,S)=∑m(0,1,3,4,8,9,15)

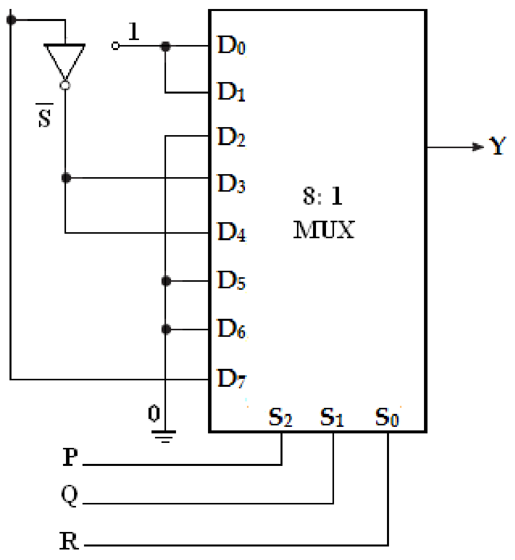
Solution:

Variables,n=4(P,Q,R,S)Sel
ectlines=n-1=3(**S2,S1,S0**)
2^{n-1}toMUXi.e.,2^3to1=8to1MUX
Inputlines=2^{n-1}=2^3=8(**D0,D1,D2,D3,D4,D5,D6,D7**)

Implementationtable:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{S}$	0	1	2	3	4	5	6	7
S	8	9	10	11	12	13	14	15
	1	1	0	$\overline{S}$	$\overline{S}$	0	0	S

Multiplexer Implementation:

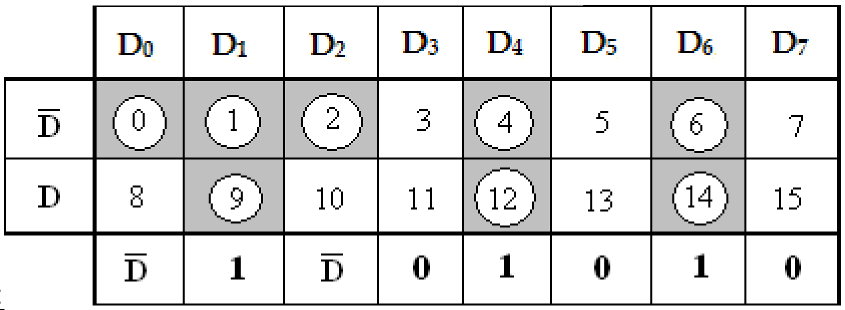


5. **ImplementtheBooleanfunctionusing8:1andalsousing4:1multiplexer F(A,B,C,D)=∑m(0,1,2,4,6,9,12,14)**

Solution:

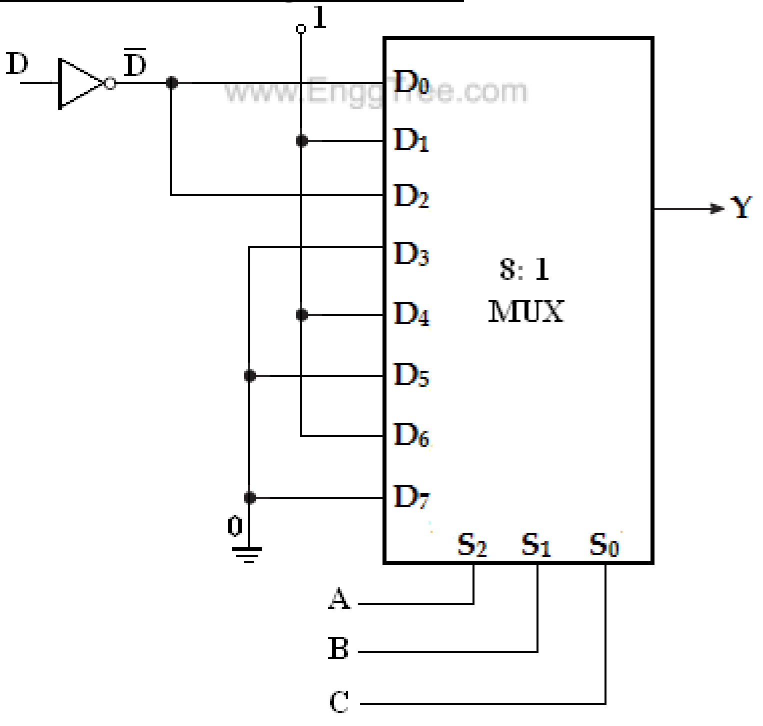
Variables,n=4(A,B,C,D)Se
lectlines=n-1=3(**S2,S1,S0**)
2^{n-1}toMUXi.e.,2^3to1=8to1MUX
Inputlines=2^{n-1}=2^3=8(**D0,D1,D2,D3,D4,D5,D6,D7**)

Implementationtable:

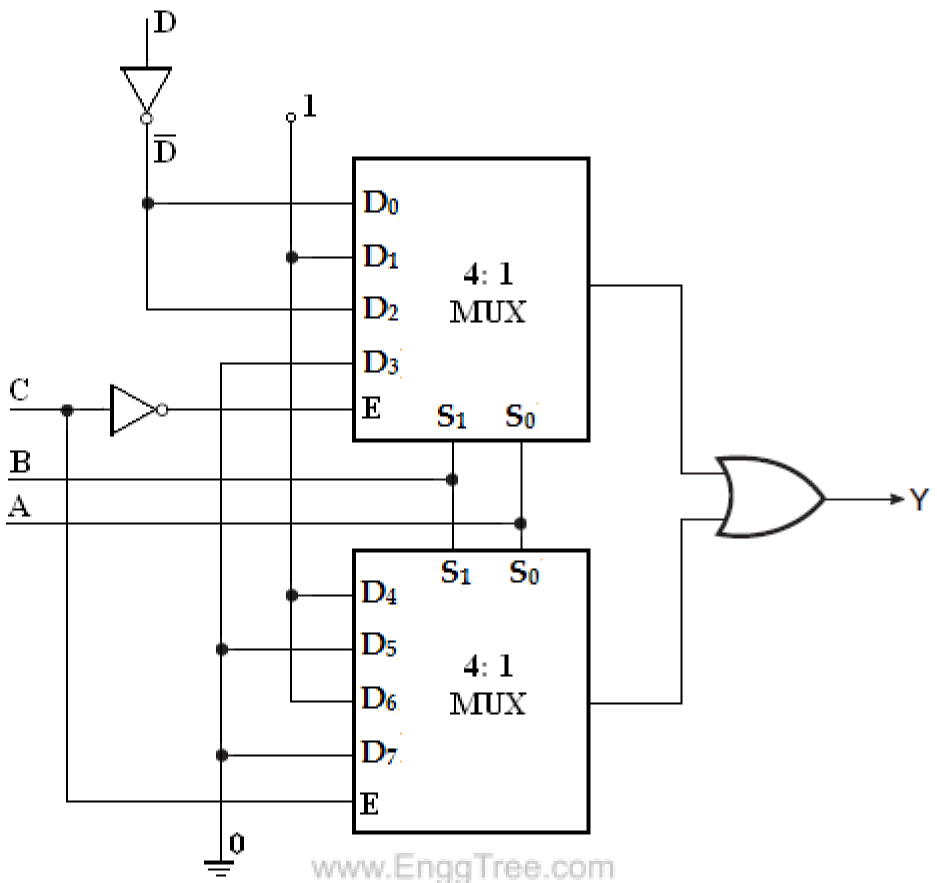


	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	$\overline{D}$	**1**	$\overline{D}$	**0**	**1**	**0**	**1**	**0**

MultiplexerImplementation(Using8:1MUX):



Using4:1MUX:



6.F(A,B,C,D)=∑m(1,3,4,11,12,13,14,15)

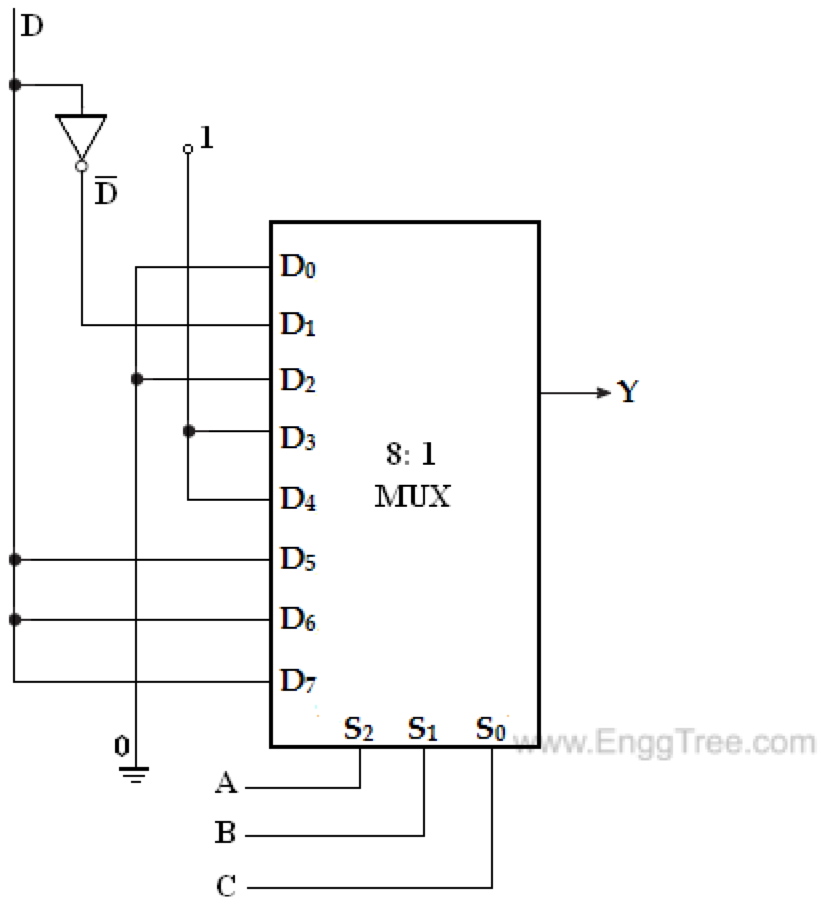
Solution:

Variables,n=4(A,B,C,D)Se
lectlines=n-1=3(**S2,S1,S0**)
2^{n-1}toMUXi.e.,2^3to1=8to1MUX
Inputlines=2^{n-1}=2^3=8(**D0,D1,D2,D3,D4,D5,D6,D7**)

Implementationtable:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	①	2	③	④	5	6	7
D	8	9	10	⑪	⑫	⑬	⑭	⑮
	0	$\overline{D}$	**0**	**1**	**1**	D	D	D

Multiplexer Implementation:



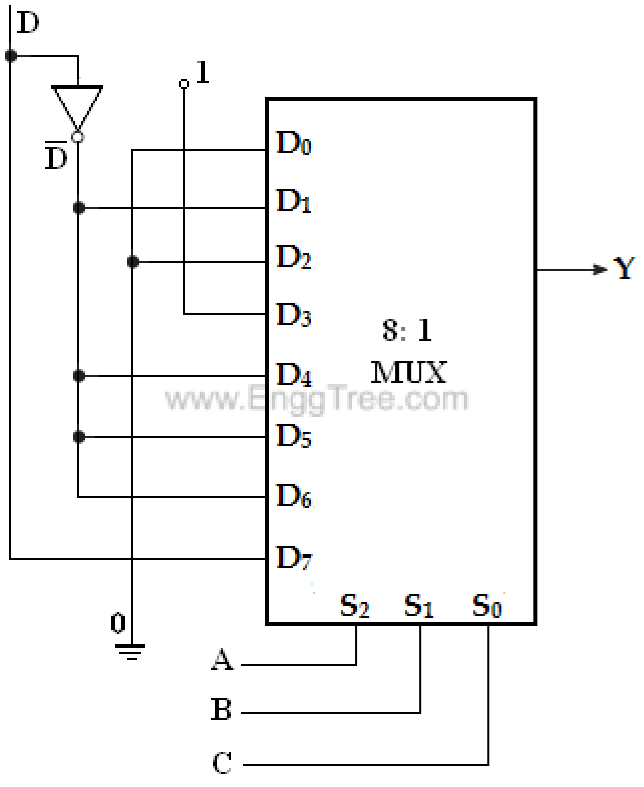
7. ImplementtheBooleanfunctionusing8:1multiplexer.
   F(A,B,C,D)=A’BD’+ACD+B’CD+A’C’D.

Solution:
ConvertintostandardSOPform,
=A‘BD‘(C‘+C)+ACD(B‘+B)+B‘CD(A‘+A)+A‘C‘D(B‘+B)
=A‘BC‘D‘+A‘BCD‘+<u>AB‘CD</u>+ABCD+A‘B‘CD+<u>AB‘CD</u>+A‘B‘C‘D
+A‘BC‘D
=A‘BC‘D‘+A‘BCD‘+AB‘CD+ABCD+A‘B‘CD+A‘B‘C‘D+A‘BC‘
D
=m4+m6+m11+m15+m3+m1+m5
=∑m (1,3,4,5,6,11,15)

Implementationtable:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{D}$	0	①	2	③	④	⑤	⑥	7
D	8	9	10	⑪	12	13	14	⑮
	0	$\bar{D}$	**0**	**1**	$\bar{D}$	$\bar{D}$	$\bar{D}$	D

Multiplexer Implementation:



8. ImplementtheBooleanfunctionusing8:1multiplexer.
   F(A,B,C,D)=AB’D+A’C’D+B’CD’+AC’D.

Solution:

ConvertintostandardSOPform,

=AB‘D(C‘+C)+A‘C‘D(B‘+B)+B‘CD‘(A‘+A)+AC‘D(B‘+B)

=AB‘C‘D+AB‘CD+A‘B‘C‘D+A‘BC‘D+A‘B‘CD‘+AB‘CD‘+AB‘C‘D+ABC‘D

=AB‘C‘D+AB‘CD+A‘B‘C‘D+A‘BC‘D+A‘B‘CD‘+AB‘CD‘+ABC‘D

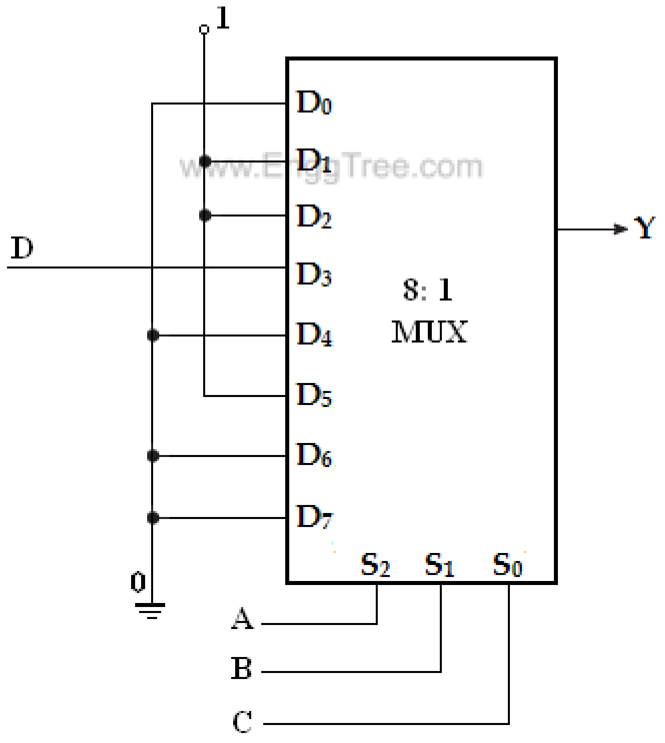
=m9+m11+m1+m5+m2+m10+m13

=∑m (1,2,5,9,10,11,13).

Implementation Table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	0	**1**	**1**	D	**0**	**1**	**0**	**0**

Multiplexer Implementation:



9. ImplementtheBooleanfunctionusing8:1andalsousing4:1multiplexer
F(w,x,y,z)=∑m(1,2,3,6,7,8,11,12,14)

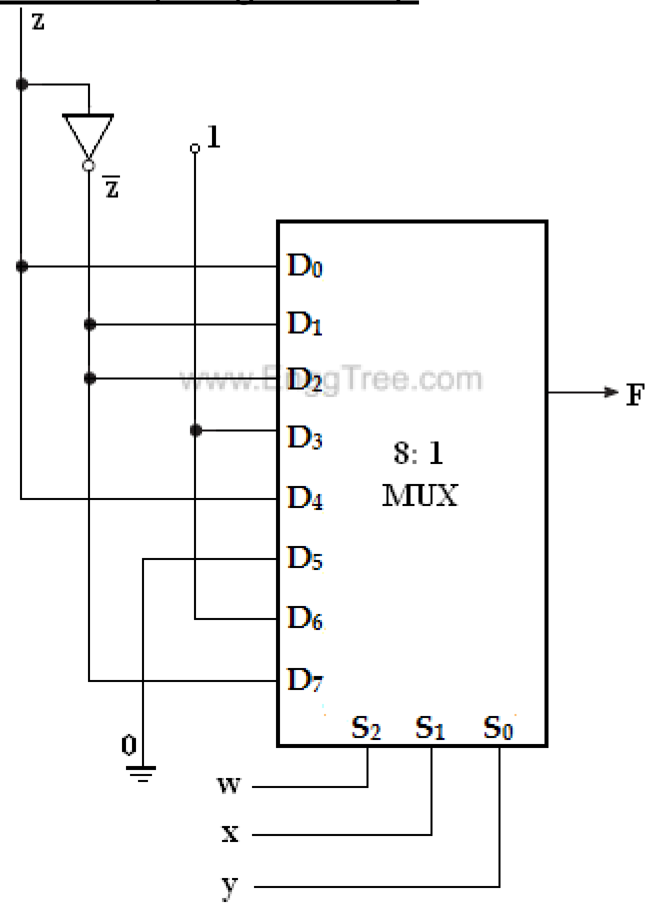
Solution:

Variables,n=4(w,x,y,z)Sele
ctlines=n-1=3(**S2,S1,S0**)
2^{n-1}toMUXi.e.,2^3to1=8to1MUX
Inputlines=2^{n-1}=2^3=8(**D0,D1,D2,D3,D4,D5,D6,D7**)

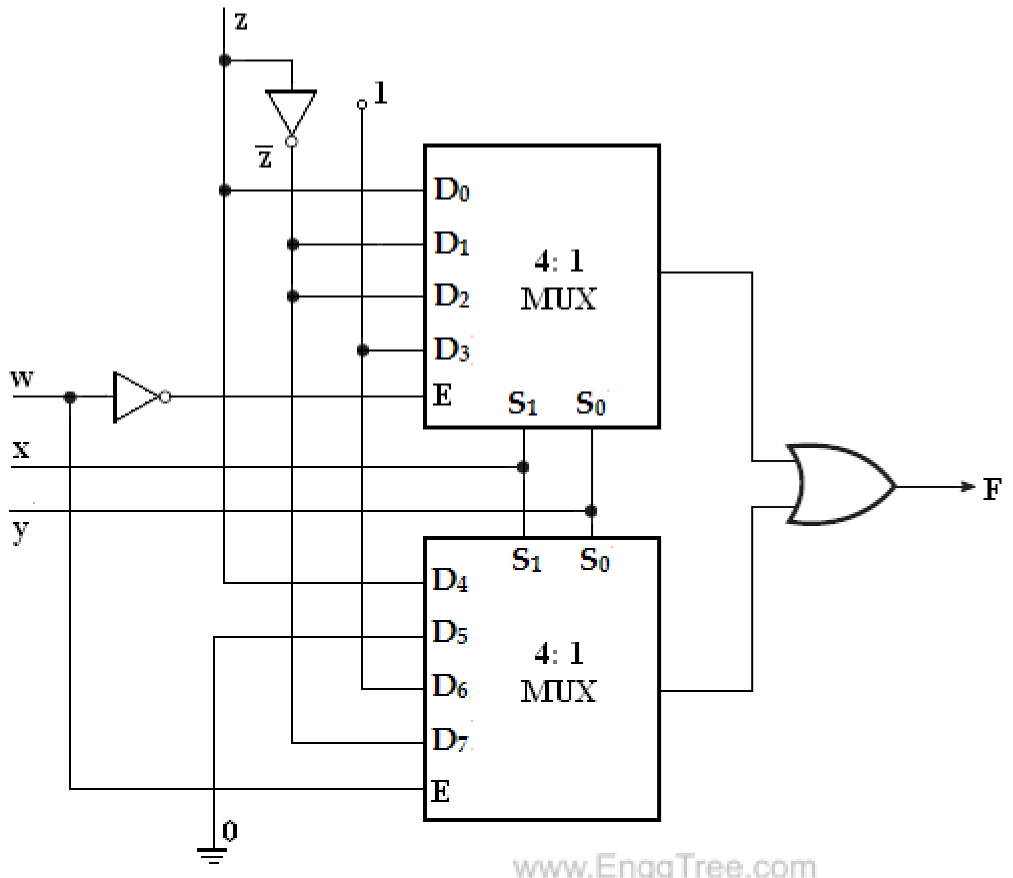
Implementationtable:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{z}$	0	①	②	③	4	5	⑥	⑦
z	⑧	9	10	⑪	⑫	13	⑭	15
	z	$\bar{z}$	$\bar{z}$	**1**	z	**0**	**1**	$\bar{z}$

MultiplexerImplementation(Using8:1MUX):



(Using4:1MUX):



10. ImplementtheBooleanfunctionusing8:1multiplexer

F(A,B,C,D)=∏m(0,3,5,8,9,10,12,14)

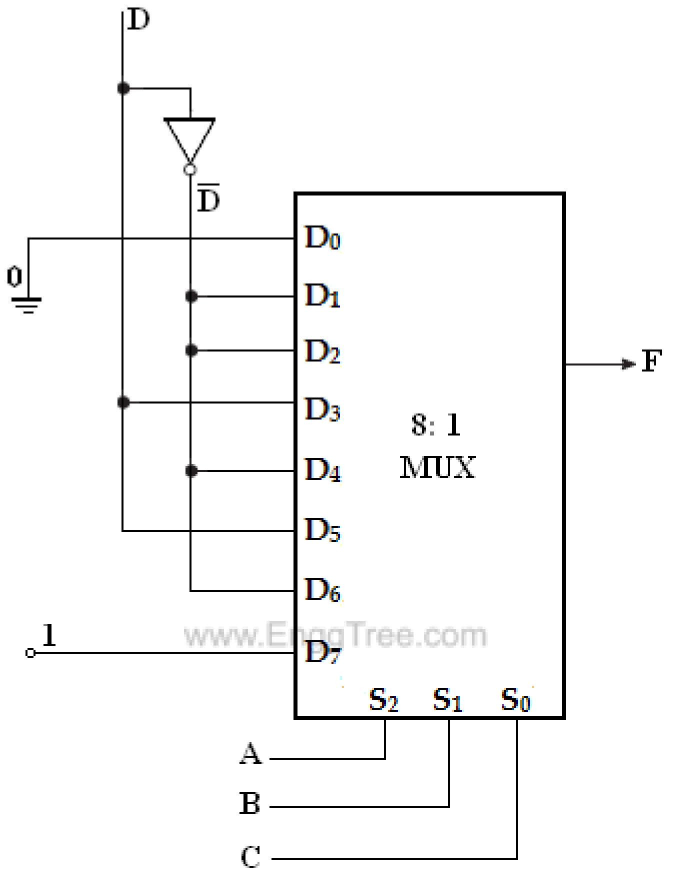
<u>Solution:</u>

Variables,n=4(A,B,C,D)Se
lectlines=n-1=3(**S2,S1,S0**)
2^{n-1}toMUXi.e.,2^3to1=8to1MUX
Inputlines=2^{n-1}=2^3=8(**D0,D1,D2,D3,D4,D5,D6,D7**)

<u>Implementationtable:</u>

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{D}$	0	①	②	3	④	5	⑥	⑦
D	8	9	10	⑪	12	⑬	14	⑮
	0	$\bar{D}$	$\bar{D}$	D	$\bar{D}$	D	$\bar{D}$	**1**

Multiplexer Implementation:



11. ImplementtheBooleanfunctionusing8:1multiplexer
F(A,B,C,D)=∑m(0,2,6,10,11,12,13)+d(3,8,14)

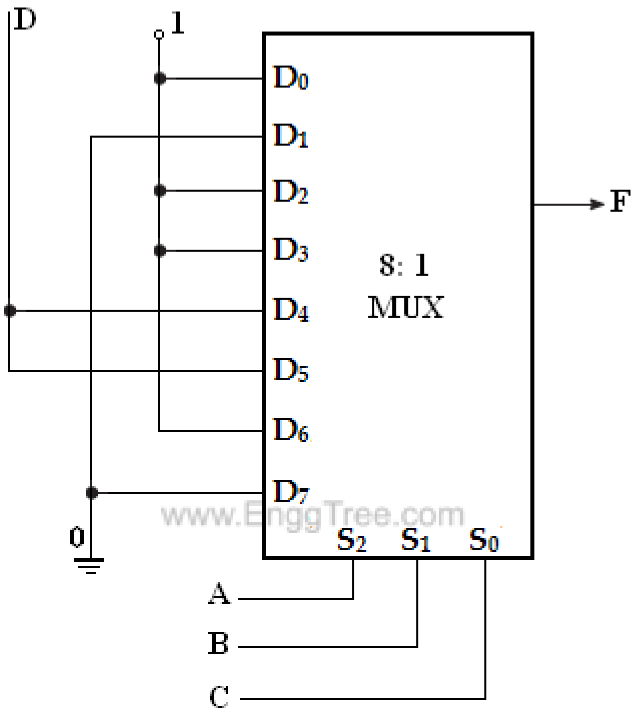
Solution:

Variables,n=4(A,B,C,D)Se
lectlines=n-1=3(**S2,S1,S0**)
2^{n-1}toMUXi.e.,2^3to1=8to1MUX
Inputlines=2^{n-1}=2^3=8(**D0,D1,D2,D3,D4,D5,D6,D7**)

ImplementationTable:

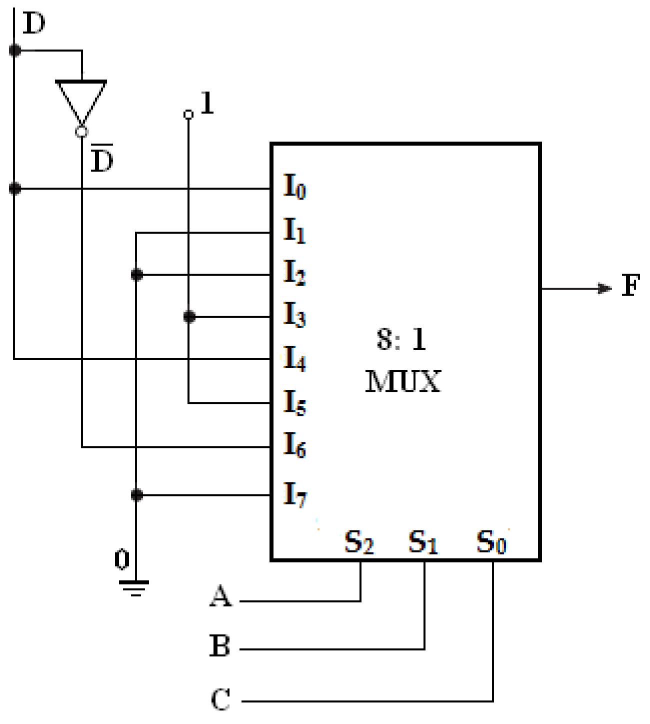
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	1	**0**	**1**	**1**	D	D	**1**	**0**

Multiplexer Implementation:



12. An8×1multiplexer hasinputsA,BandC connectedtotheselectioninputsS2,S1,andS0respectively.ThedatainputsI0t o I7areasfollows **I1=I2=I7=0;I3=I5=1;I0=I4=D**and**I6=D'.** DeterminetheBooleanfunctionthatthemultiplexerimplements.

Multiplexer Implementation:



Implementationtable:

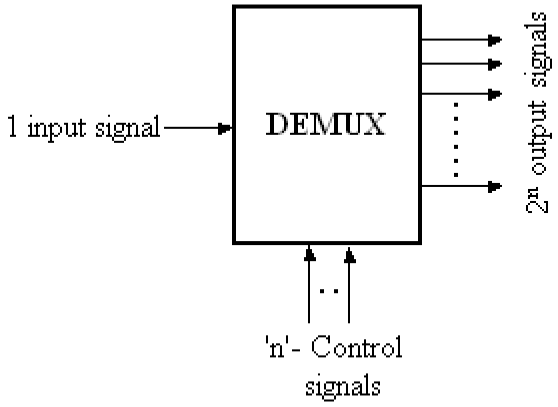
	I_0	I_1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	D	0	0	1	D	1	$\overline{D}$	0

F(A,B,C,D)=∑m(3,5,6,8,11,12,13).

DEMULTIPLEXER:

Demultiplexmeansoneintomany.Demultiplexingistheprocessoftakinginformationfromoneinputandtransmittingthesameoveroneofseveraloutputs.

Ademultiplexerisacombinationallogiccircuitthatreceivesinformationonasingleinputandtransmitsthesameinformationoveroneofseveral(2^n)outputlines.

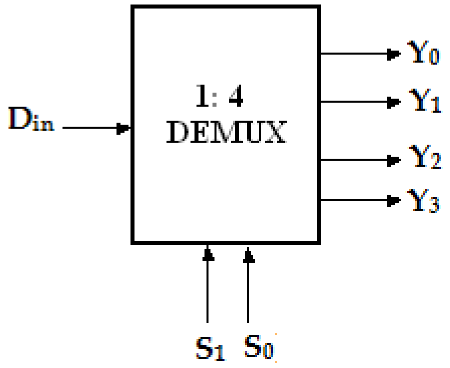


Blockdiagramofdemultiplexer

The block diagram of a demultiplexer which is opposite to a multiplexer in itsoperation is shown above. The circuit has one input signal, ‗n' select signals and 2^noutput signals. The select inputs determine to which output the data input will beconnected. As the serial data is changed to parallel data, i.e., the input caused to appearononeofthenoutputlines,thedemultiplexerisalsocalleda―***datadistributer***‖ ora
―***serial-to-parallelconverter***‖.

1-to-4Demultiplexer:

A1-to-



4demultiplexerhasasingleinput,**D_{in}**,fouroutputs(**Y_0toY$_3$**)andtwoselectinputs(**S_1andS$_0$**).

LogicSymbol

The input variable Dinhas a path to all four outputs, but the input information isdirected to only one of the output lines. The truth table of the 1-to-4 demultiplexer isshownbelow.

Enabl e	S1	S0	Din	Y0	Y1	Y2	Y3
0	x	x	x	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

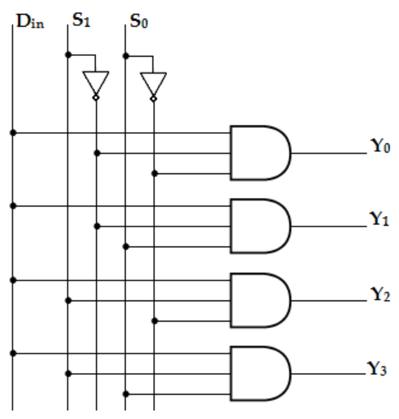
Truthtableof1-to-4demultiplexer

Fromthetruthtable,itisclearthatthedatainput,DinisconnectedtotheoutputY_0, when S_1= 0 and S_0= 0 and the data input is connected to output Y_1 when S_1= 0 andS_0= 1. Similarly, the data input is connected to output $Y_2$and $Y_3$when S_1= 1 and S_0= 0and when S_1= 1 and S_0= 1, respectively. Also, from the truth table, the expression foroutputscanbewrittenasfollows,

Y0=

S1'S0'DinY1=

S1'S0DinY2=S1S0



'Din

Y3=S1S0Din

Logicdiagramof1-to-4demultiplexer

Now, using the above expressions, a 1-to-4 demultiplexer can be implementedusingfour3-inputAND gatesand twoNOT gates. Here,theinputdatalineDin,isconnectedto allthe ANDgates.ThetwoselectlinesS1,S0enableonlyonegate at a time.andthedatathatappearsontheinputlinepassesthroughtheselectedgatetotheassoci atedoutputline.

<u>1-to-8Demultiplexer</u>:

A1-to-8demultiplexerhasasingleinput,**Din**,eightoutputs(**Y0toY7**)andthreeselectinputs(**S2,S1andS0**).Itdistributesoneinputlinetoeightoutputlinesbasedontheselectinputs.Thetruthtableof1-to-8demultiplexerisshownbelow.

Din	S2	S1	S0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

<u>Truthtableof1-to-8demultiplexer</u>

Fromtheabovetruthtable,itisclearthatthedatainputisconnectedwithoneoftheeightoutputsbasedontheselectinputs.Nowfromthistruthtable,theexpressionforeightoutputscanbewrittenasfollows:

Y0=S2‘S1‘S0‘Din

Y1=S2‘S1‘S0Din

Y2=S2‘S1S0‘Din

Y3=S2‘S1S0Din

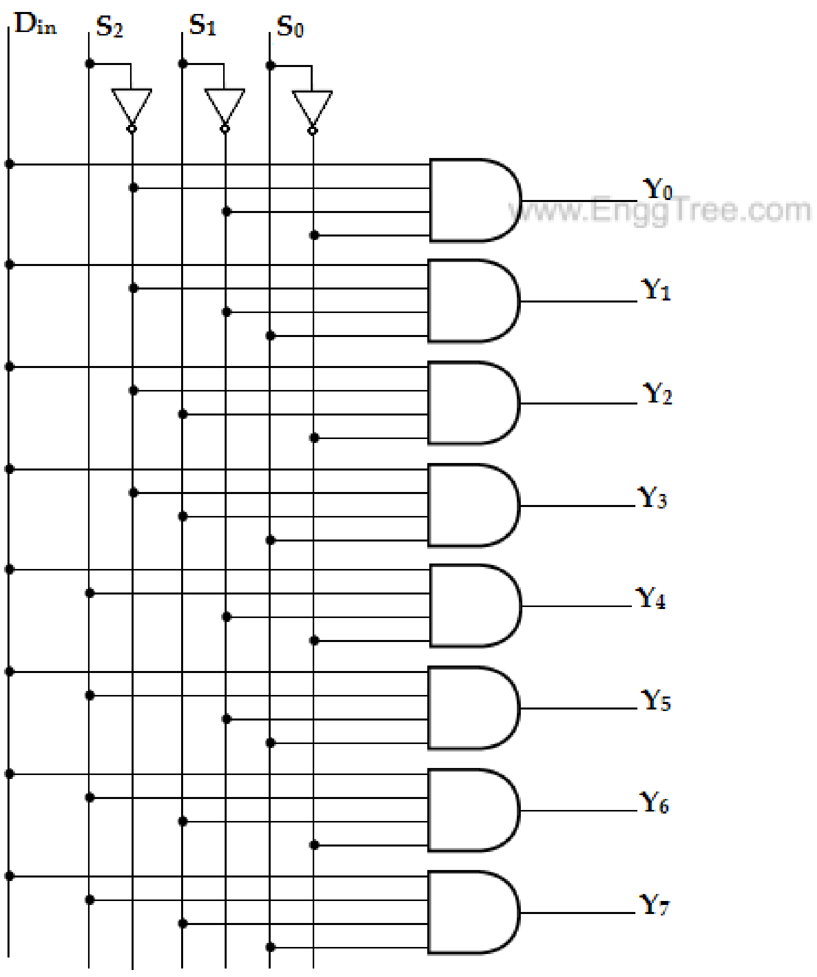
Y4= S2S1‘S0‘Din

Y5= S2S1‘S0Din

Y6= S2S1S0‘Din

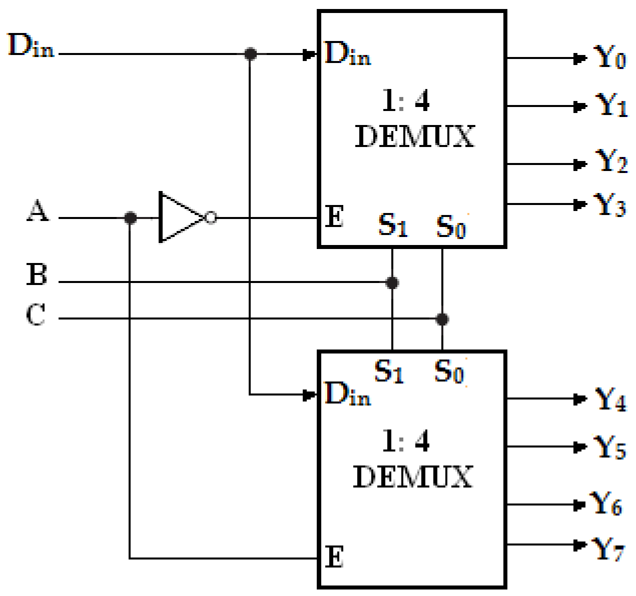
Y7=S2S1S0Din

Now using the above expressions, the logic diagram of a 1-to-8 demultiplexer can bedrawn as shown below. Here, the single data line, Din is connected to all the eight ANDgates, but only one of the eight AND gates will be enabled by the select input lines. Forexample, if S2S1S0= 000, then only AND gate-0 will be enabled and thereby the datainput,DinwillappearatY0.Similarly,thedifferentcombinationsoftheselectinputs,theinputDinwillappearattherespectiveoutput.



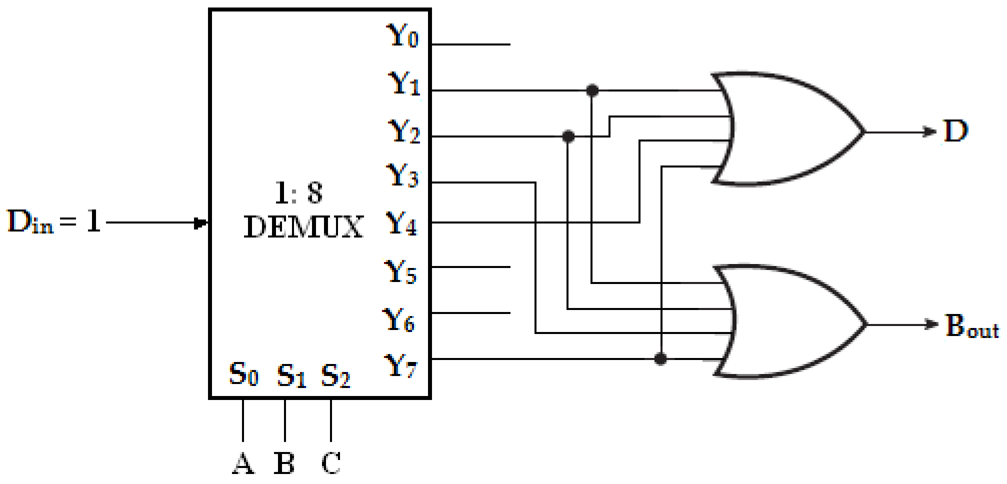
Logicdiagramof1-to-8demultiplexer

1. Design1:8demultiplexerusingtwo1:4DEMUX.



2. Implementfullsubtractorusingdemultiplexer.

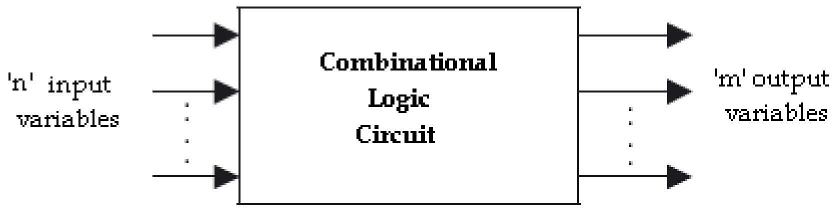
Inputs			**Outputs**	
A	**B**	**Bin**	**Difference(D)**	**Borrow(Bout)**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



UNITII-SYNCHRONOUS SEQUENTIAL LOGIC

INTRODUCTION

In*combinationallogiccircuits*,theoutputsatanyinstantoftimedependonlyontheinput signalspresentatthattime.Forachangeininput,theoutputoccursimmediately.

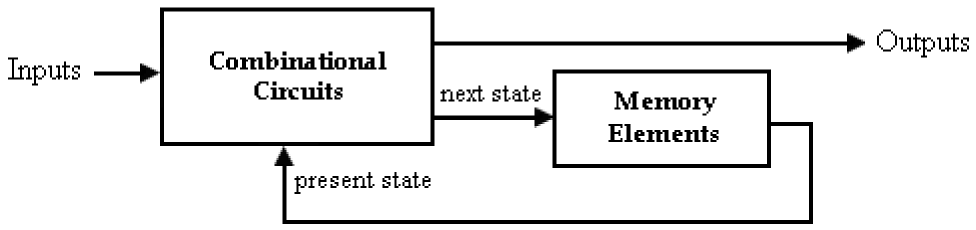


CombinationalCircuit-BlockDiagram

In ***sequential logic circuits***, it consists of combinational circuits to whichstorage elements are connected to form a feedback path. The storage elements aredevicescapableofstoringbinaryinformationeither1or0.

The information stored in the memory elements at any given time defines thepresentstateofthesequentialcircuit.Thepresentstateandtheexternalcircuitdeterminetheo utputandthenextstateofsequentialcircuits.

SequentialCircuit-BlockDiagram

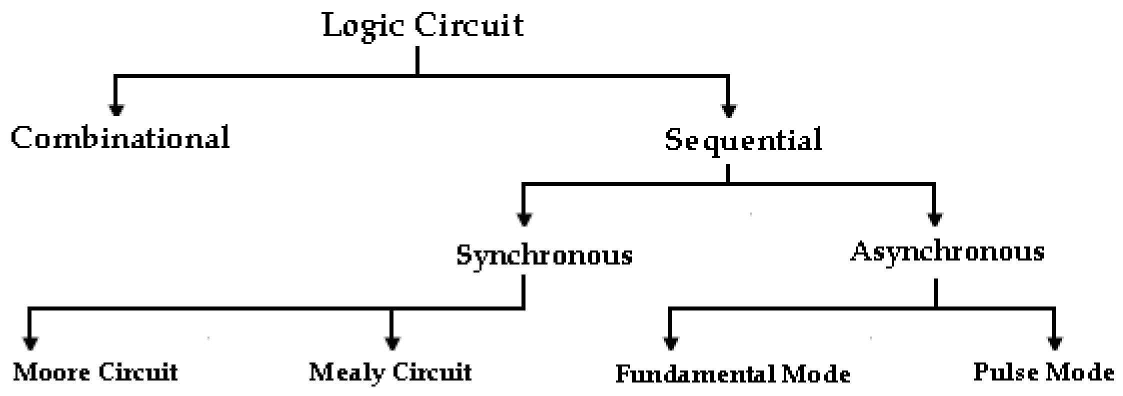


Thusinsequentialcircuits,theoutputvariablesdependnotonlyonthepresent input variablesbut also on the past historyof input variables.

Therotarychannelselectedknobonanold-fashionedTVislikeacombinational. Its output selects a channel based only on its current input – theposition of the knob. The channel-up and channel-down push buttons on a TV is likeasequentialcircuit.Thechannelselectiondependsonthepastsequenceofup/downpus hes.Thecomparisonbetweencombinationalandsequentialcircuitsisgivenintablebelow.

S.No	Combinationallogic	Sequentiallogic
1	Theoutputvariable,atalltimesd ependsonthecombinationof inputvariables.	Theoutputvariabledependsnotonlyo nthepresentinputbutalsodepend uponthepasthistoryofinputs.
2	Memoryunitisnotrequired	Memoryunitisrequiredtostorethe pasthistoryofinputvariables.
3	Fasterinspeed	Slowerthancombinationalcircuits.
4	Easy to design	Comparativelyhardertodesign.
5	Eg.Paralleladder	Eg.Serialadder

ClassificationofLogicCircuits



Thesequentialcircuitscanbeclassifieddependingonthetimingoftheirsignals:

- Synchronoussequentialcircuits
- Asynchronoussequentialcircuits.

In synchronous sequential circuits, signals can affect the memory elementsonlyatdiscreteinstantsoftime.Inasynchronoussequentialcircuitschange ininput signals can affect memory element at any instant of time. The memoryelements used in both circuits are Flip-Flops, which are capable of storing 1-bitinformation.

S.No	Synchronoussequentialcircuits	Asynchronoussequentialcircuits
1	Memoryelements are clocked Flip-Flops	Memoryelementsareeitherunclocked Flip-Flopsortimedelayelements.
2	Thechangeininputsignalscan affect memory element uponactivationofclocksignal.	Thechangeininputsignalscanaffectmemoryelementatanyinstantoftime.
3	Themaximumoperatingspeedofclockdependsontimedelays involved.	Becauseoftheabsenceofclock,itcanoperate faster than synchronous circuits.
4	Easiertodesign	Moredifficulttodesign

LATCHES:

Latches and Flip-Flops are the basic building blocks of the most sequentialcircuits.Latchesareusedforasequentialdevicethatchecksallofitsinputscontinuouslyandchangesitsoutputsaccordinglyatanytimeindependentofclockingsignal.Enablesignalisprovidedwiththelatch.Whenenablesignal isactive output changes occur as the input changes. But when enable signal is notactivatedinputchangesdonotaffecttheoutput.

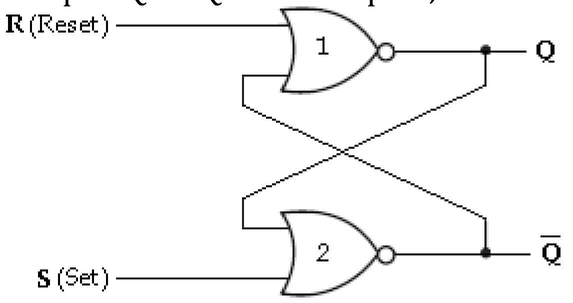
Flip-Flop is used for a sequential device that normally samples its inputs andchangesitsoutputsonlyattimesdeterminedbyclockingsignal.

SRLatch:

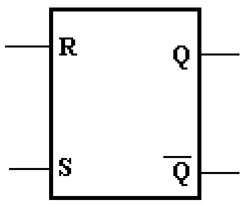
The simplest type of latch is the set-reset (SR) latch. It can be constructed fromeithertwoNORgatesortwoNANDgates.

SRlatchusingNORgates:

The two NOR gates are cross-coupled so that the output of NOR gate 1 isconnected to one of the inputs of NOR gate 2 and vice versa. The latch has twooutputsQandQ'andtwoinputs,setandreset.



SRlatchusingNOR gates



LogicSymbol

Before going to analyse the SR latch, we recall that a logic 1 at any input of aNORgateforcesitsoutputtoalogic0.Letusunderstandtheoperationofthiscircuitforvariou sinput/outputpossibilities.

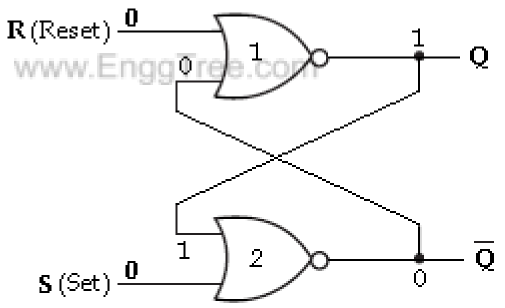
Case1:S=0andR=0

Initially,Q=1andQ'=0

Letus assumethatinitiallyQ=1and Q'=0. WithQ'=0,both inputs to NORgate 1 are at logic 0. So, its output, Q is at logic 1. With Q=1, one input of NOR gate 2isatlogic

1. Hence its output, Q' is at logic 0. This shows that when S and R both arelow,theoutputdoesnotchange.

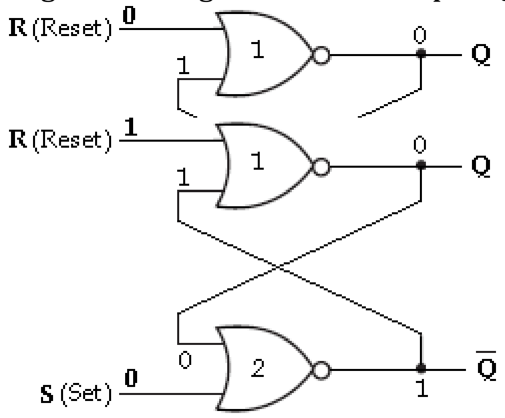
Initially,Q=0andQ'=1



WithQ'=1,oneinputofNORgate1isatlogic1,henceitsoutput,Qisatlogic 0.WithQ=0,bothinputstoNORgate2areatlogic 0.So,itsoutputQ'isatlogic1.Inthiscasealsothereisnochangeintheoutputstate.

Case2:S=0andR=1

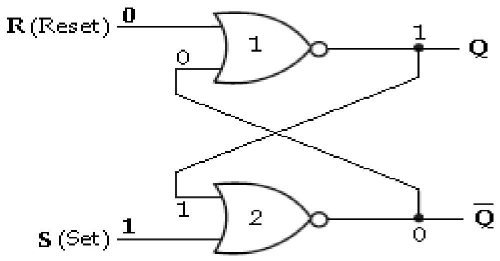
Inthiscase,RinputoftheNORgate1isatlogic1,henceitsoutput,Qisatlogic0.BothinputstoNO



Rgate2arenowatlogic 0.Sothatitsoutput,Q'is atlogic 1.

Case3:S=1andR=0

In thiscase,Sinputofthe NORgate 2isat logic1, henceitsoutput,Q isat logic0.Bothinputs toNORgate1arenow atlogic0.Sothatits output,Qisatlogic 1.



Case4:S=1andR=1

When R and S both are at logic 1, they force the outputs of both NOR gates tothe low state, i.e., (Q=0 and Q'=0). So, we call this an indeterminate or prohibitedstate,andrepresentthisconditioninthetruthtableasanasterisk(*).Thisconditionalso violates the basic definition of a latch that requires Q to be complement of Q'.Thusinnormaloperationthisconditionmustbeavoidedbymakingsurethat1'sarenotappliedtoboththeinputssimultaneously.

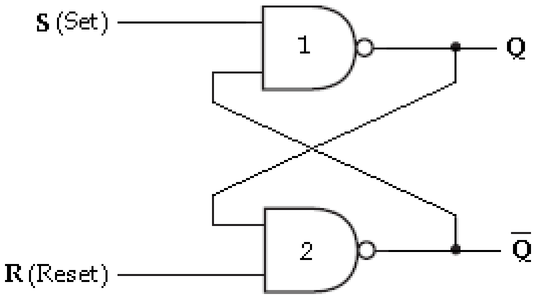
WecansummarizetheoperationofSRlatchasfollows:

- WhenS=0andR= 0,theoutput, Q_{n+1}remainsinitspresentstate,Q_n.
- WhenS=0andR=1,thelatchisresetto0.
- WhenS=1andR=0,thelatchissetto1.
- WhenS=1andR=1,theoutputofbothgateswillproduce0.i.e.,**Q_{n+1}= Q_{n+1}'=0.**

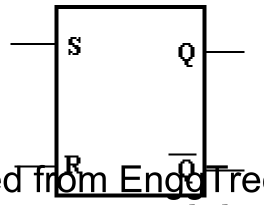
S	R	Q_n	Q_{n+1}	State
0 0	0 0	0 1	0 1	No Change (NC)
0 0	1 1	0 1	0 0	Reset
1 1	0 0	0 1	1 1	Set
1 1	1 1	0 1	x x	Indeterminate *

<u>SRlatchusingNANDgates:</u>

TheSRlatchcanalsobeimplementedusingNANDgates.TheinputsofthisLatchareSandR.Tounderstandhowthiscircuitfunctions,recallthatalowonanyinputtoaNANDgateforcesitsoutputhigh.



<u>SRlatchusingNANDgates</u>



<u>LogicSymbol</u>

WecansummarizetheoperationofSRlatchasfollows:

- WhenS=0andR=0,theoutputofboth gateswillproduce0.i.e.,**Q_{n+1}=Q_{n+1}'=1.**
- WhenS=0andR=1,thelatchisresetto0.
- WhenS=1andR=0,thelatchissetto1.
- WhenS=1andR= 1,theoutput, Q_{n+1}remainsinitspresentstate,Q_n.

S	R	Q_n	Q_{n+1}	State
0 0	0 0	0 1	x x	Indeterminate *
0 0	1 1	0 1	1 1	Set
1 1	0 0	0 1	0 0	Reset
1 1	1 1	0 1	0 1	No Change (NC)

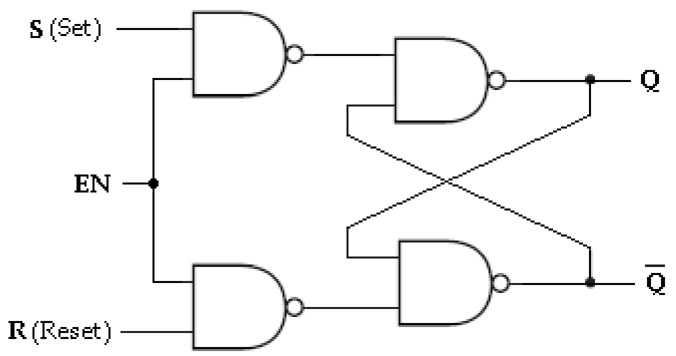
GatedSRLatch:

IntheSRlatch,theoutputchangesoccurimmediatelyaftertheinputchangesi.e,thelatchissensitivetoitsSandRinputsallthetime.

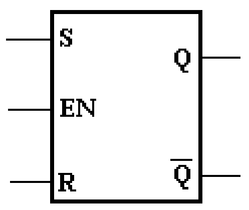
Alatchthatissensitivetotheinputsonlywhenanenableinputisactive.SuchalatchwithenableinputisknownasgatedSRlatch.

- ThecircuitbehaveslikeSRlatchwhenEN=1.ItretainsitspreviousstatewhenEN =0

SRLatchwithenableinputusingNANDgates



LogicSymbol

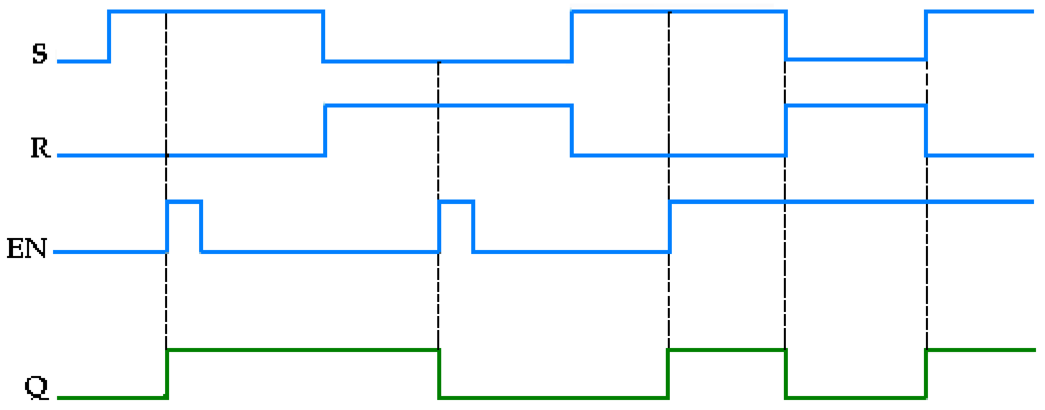


ThetruthtableofgatedSRlatchisshowbelow.

EN	S	R	Q_n	Q_{n+1}	State
1 1	0 0	0 0	0 1	0 1	No Change(NC)
1 1	0 0	1 1	0 1	0 0	Reset
1 1	1 1	0 0	0 1	1 1	Set
1 1	1 1	1 1	0 1	x x	Indeterminate *

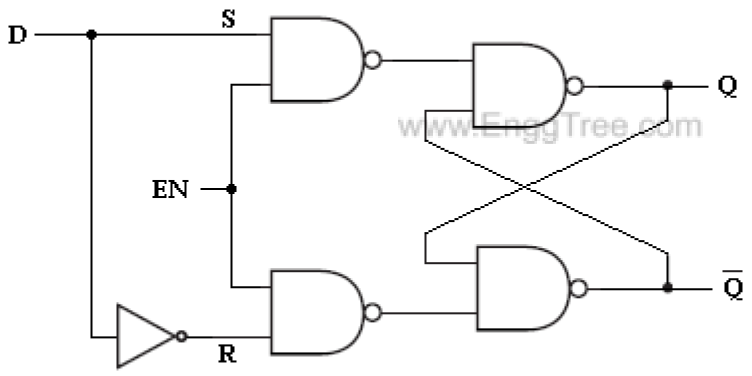
0	x	x	0	0	No Change(NC)
0	x	x	1	1	

WhenSisHIGHandRisLOW,aHIGHontheENinputsetsthelatch.WhenSisLOWandRisHIGH, aHIGHontheENinputresetsthelatch.

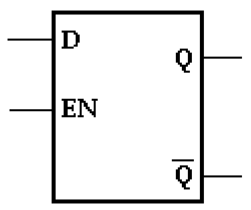


DLatch

In SR latch, when both inputs are same (00 or 11), the output either does notchange or it is invalid. In many practical applications, these input conditions are notrequired. These input conditions can be avoided by making them complement ofeachother.ThismodifiedSRlatchisknownas**Dlatch.**



DLatch



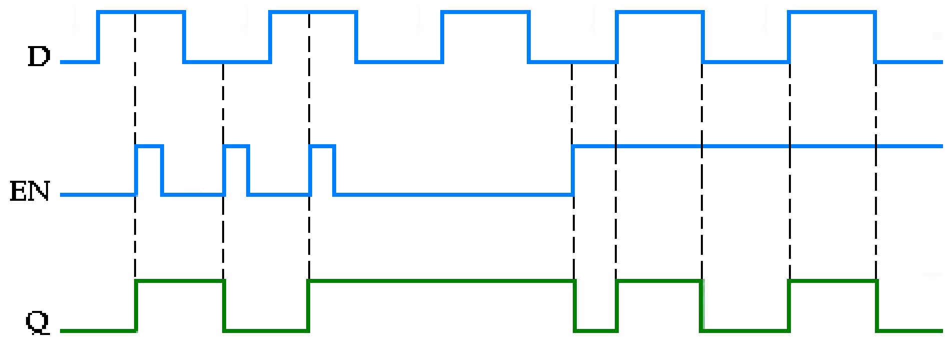
LogicSymbol

Asshowninthefigure,DinputgoesdirectlytotheSinput,anditscomplement is applied to the R input. Therefore, only two input conditions exists,eitherS=0andR=1orS=1andR=0.The truthtableforDlatchisshownbelow.

EN	D	Q_n	Q_{n+1}	State
1	0	x	0	Reset
1	1	x	1	Set
0	x	x	Q_n	No Change(NC)

Asshowninthetruthtable,theQoutputfollowstheDinput.Forthisreason,Dlatchiscalled*transparentlatch*.

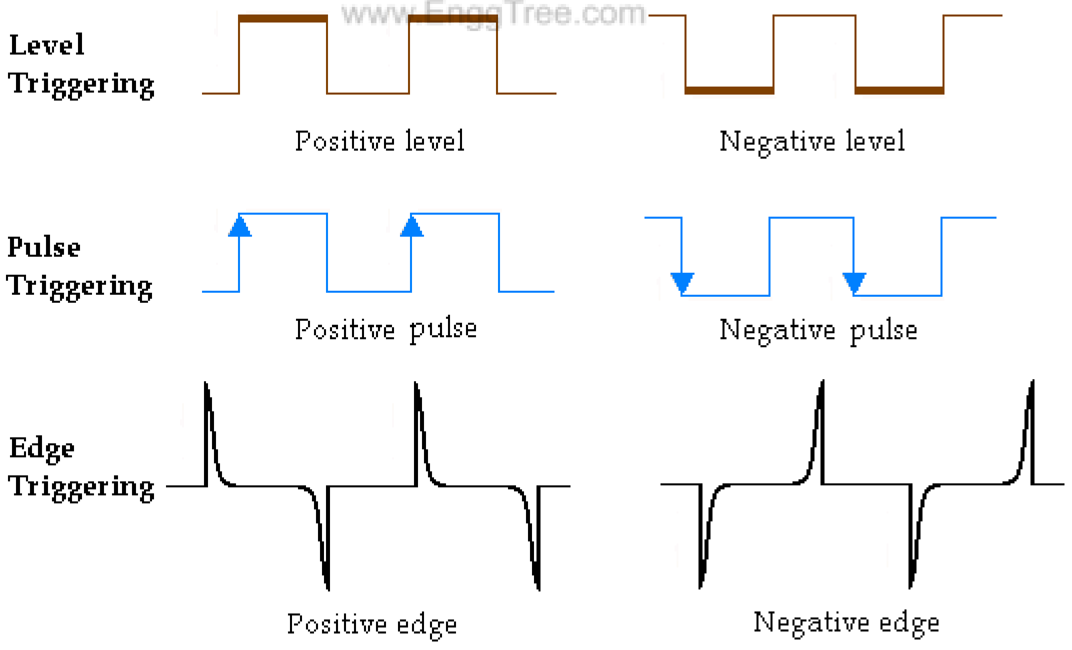
When D is HIGH and EN is HIGH. Q goes HIGH. When D is LOW and EN isHIGH,QgoesLOW.WhenENisLOW,thestateofthelatchisnotaffectedbytheDinput.



TRIGGERINGOFFLIP-FLOPS

ThestateofaFlip-Flopisswitchedbyamomentarychangeintheinputsignal.This momentary change is called a trigger and the transition it causes is said totrigger the Flip-Flop. Clocked Flip-Flops are triggered by pulses. A clock pulse startsfrom an initial value of 0, goes momentarily to 1and after a short time, returns to itsinitial0value.

Latches are controlled by enable signal, and they are level triggered, eitherpositive level triggered or negative level triggered. The output is free to changeaccording to the S and R input values, when active level is maintained at the enableinput.Flip-Flopsaredifferentfromlatches.Flip-Flopsarepulseorclockedgetriggeredinsteadofleveltriggered.



EDGETRIGGEREDFLIP-FLOPS

Flip-Flops are synchronous bistable devices (has two outputs Q and Q’). Inthiscase,thetermsynchronousmeansthattheoutputchangesstateonlyataspecified point on the triggering input called the clock (CLK), i.e., changes in theoutputoccurinsynchronizationwiththeclock.

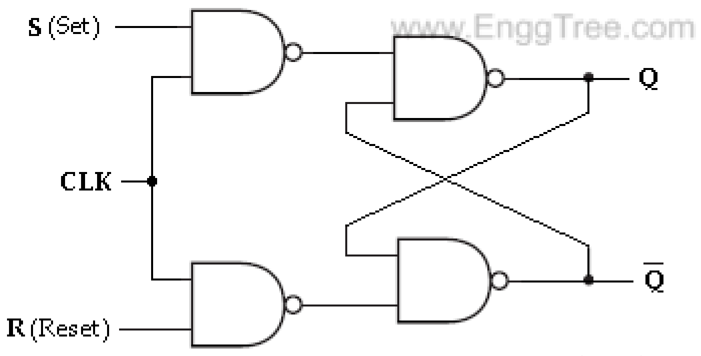
An*edge-triggeredFlip-Flop*changesstateeitheratthepositiveedge(rising edge) or at the negative edge (falling edge) of the clock pulse and is sensitive to itsinputs only at this transition of the clock. The different types of edge-triggered Flip-Flopsare—

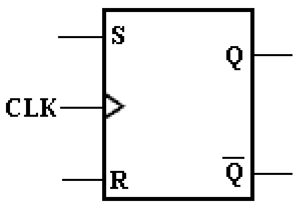
- S-RFlip-Flop,
- J-KFlip-Flop,
- DFlip-Flop,
- TFlip-Flop.

Although the S-R Flip-Flop is not available in IC form, it is the basis for the DandJ-KFlip-Flops.Eachtypecanbeeitherpositiveedge-triggered(nobubbleatC input) or negative edge-triggered (bubble at C input). The key to identifying anedge-triggered Flip-Flop by its logic symbol is the small triangle inside the block attheclock(C)input.Thistriangleiscalledthe**dynamicinputindicator**.

S-RFlip-Flop

TheSandRinputsoftheS-RFlip-Floparecalled*synchronous*inputsbecausedata on these inputs are transferred to the Flip-Flop's output only on the triggeringedge of the clock pulse. The circuit is similar to SR latch except enable signal isreplacedbyclockpulse(CLK).Onthepositiveedgeoftheclockpulse,thecircuit respondstotheSandRinputs.





Logic Symbol

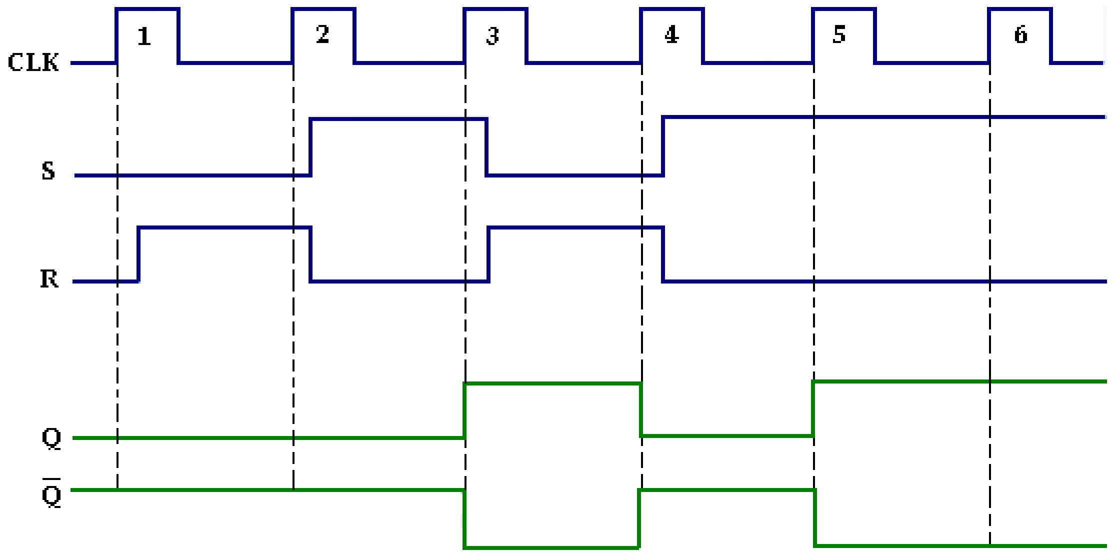
<u>**SRFlip-Flop**</u>

WhenSisHIGHandRisLOW,theQoutputgoesHIGHonthetriggeringedgeoftheclockpulse,andtheFlip-FlopisSET.WhenSisLOWandRisHIGH,theQ output goes LOW on the triggering edge of the clock pulse, and the Flip-Flop isRESET. When both S and R are LOW, the output does not change from its prior state.AninvalidconditionexistswhenbothSandRareHIGH.

CLK	S	R	Q_n	Q_{n+1}	State
1 1	0 0	0 0	0 1	0 1	No Change(NC)
1 1	0 0	1 1	0 1	0 0	Reset
1 1	1 1	0 0	0 1	1 1	Set

1 1	1 1	1 1	0 1	x x	Indeterminate *
0 0	x x	x x	0 1	0 1	No Change(NC)

TruthtableforSRFlip-Flop



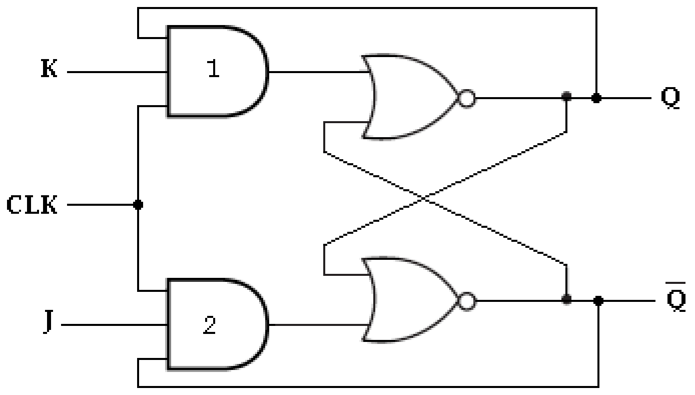
InputandoutputwaveformsofSRFlip-Flop

J-KFlip-Flop:

JKmeansJackKilby,TexasInstrument(TI)Engineer,whoinventedICin1958.JK Flip-Flop has two inputs J(set) and K(reset). A JK Flip-Flop can be obtained fromtheclockedSRFlip-FlopbyaugmentingtwoANDgatesasshownbelow.

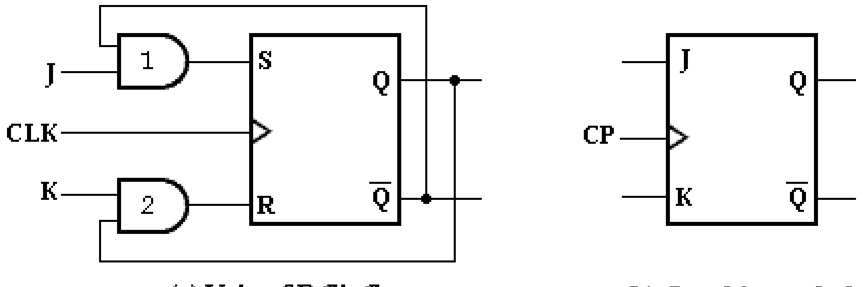
JKFlipFlop

The data input J and the output Q' are applied o the first AND gate and



itsoutput(JQ')isappliedtotheSinputofSRFlip-Flop.Similarly,thedatainputKand theoutputQareappliedtothesecondANDgateanditsoutput(KQ)isappliedtotheRinputof SRFlip-Flop.

J=K=0



(a) Using SR flipflop (b) Graphic symbol

WhenJ=K=0,bothANDgatesaredisabled.Thereforeclockpulsehavenoeffect,henc etheFlip-Flopoutputis sameasthepreviousoutput.

J=0,K=1

WhenJ=0andK=1,ANDgate1isdisabledi.e.,S=0andR=1.Thisconditionwillresetth eFlip-Flopto0.

J=1,K=0

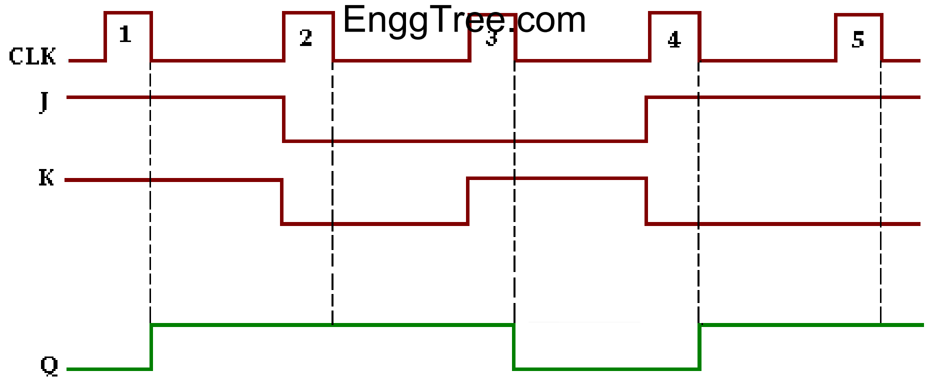
WhenJ=1andK=0,ANDgate2isdisabledi.e.,S=1andR=0.ThereforetheFlip-Flopwillsetontheapplicationofaclockpulse.

J=K=0

WhenJ=K=1,itispossibletosetorresettheFlip-Flop.IfQisHigh,ANDgate 2 passes on a reset pulse to the next clock. When Q is low, AND gate 1 passes ona set pulse to the next clock. Eitherway, Q changes to the complement of the laststatei.e.,toggle.Togglemeanstoswitchtotheoppositestate.

Thetruth tableof JKFlip-Flop isgiven below.

CLK	Inputs		Output	State
	J	K	Q_{n+1}	
1	0	0	Q_n	No Change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	Q_n'	Toggle



InputandoutputwaveformsofJKFlip-Flop

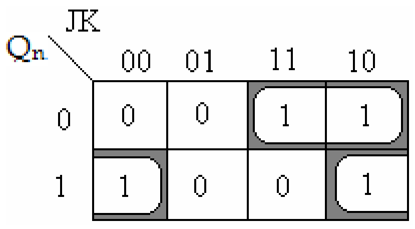
CharacteristictableandCharacteristicequation:

The characteristic table for JK Flip-Flop is shown in the table below. From thetable, K-map for the next state transition (Q_{n+1}) can be drawn and the simplified logicexpressionwhichrepresentsthecharacteristicequationofJKFlip-Flopcanbefound.

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Characteristictable

K-mapSimplification:

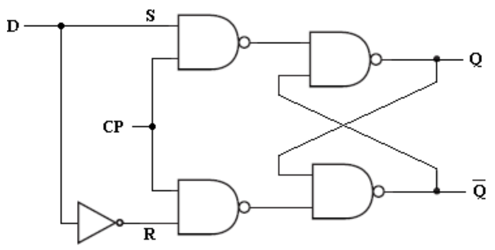


Characteristicequation:$\mathbf{Q_{n+1}=JQ'+K'Q}$.

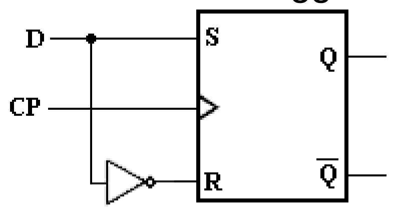
DFlip-Flop:

LikeinDlatch,inDFlip-FlopthebasicSRFlip-Flopisusedwithcomplementedinputs.TheDFlip-FlopissimilartoD-latchexceptclockpulseisusedinsteadofenableinput.

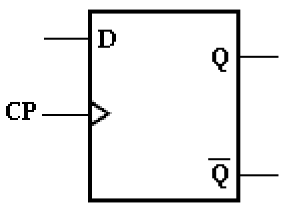
DFlip-Flop



ToeliminatetheundesirableconditionoftheindeterminatestateintheRSFlip-FlopistoensurethatinputsSandRareneverequalto1atthesametime.Thisis donebyD Flip-Flop.TheD (*delay*) Flip-Flophas one inputcalled delayinputandclockpulseinput.TheDFlip-FlopusingSRFlip-Flopisshownbelow.



(a) Using SR flipflop

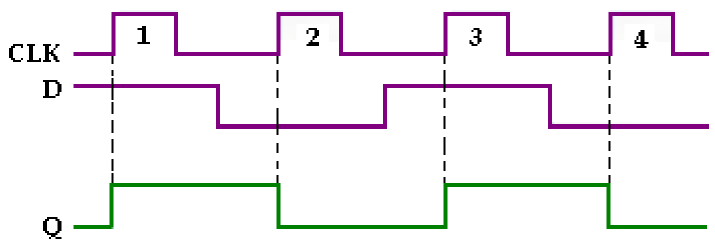


(b) Graphic symbol

The truth table of D Flip-Flop isgiven below.

Clock	D	Q_{n+1}	State
1	0	0	Reset
1	1	1	Set
0	x	Q_n	No Change

TruthtableforDFlip-Flop



InputandoutputwaveformsofclockedDFlip-Flop

Looking at the truth table for D Flip-Flop we can realize that Q_{n+1} functionfollowstheD inputatthepositivegoingedgesoftheclockpulses.

CharacteristictableandCharacteristicequation:

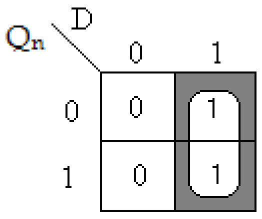
ThecharacteristictableforDFlip-Flopshows thatthenextstateof theFlip-FlopisindependentofthepresentstatesinceQ_{n+1}isequaltoD.Thismeansthataninputpulsewilltransferthevalueofinput Dintotheoutputofthe Flip-Flopindependentofthevalueoftheoutputbeforethepulsewasapplied.

ThecharacteristicequationisderivedfromK-map.

Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

Characteristictable

K-map simplification



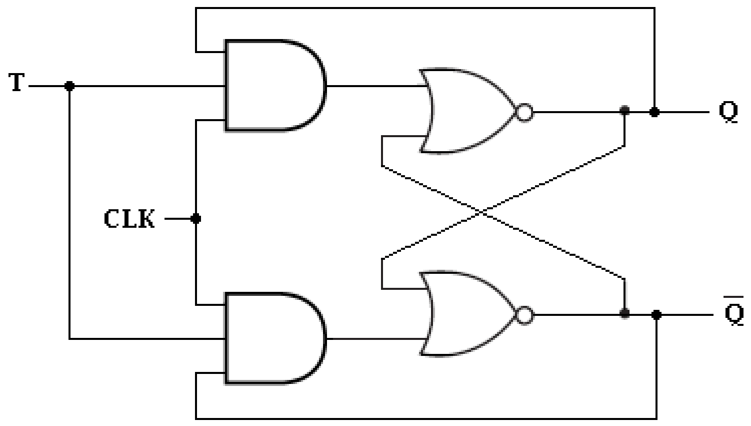
Characteristicequation:**Q_{n+1}=D.**

TFlip-Flop

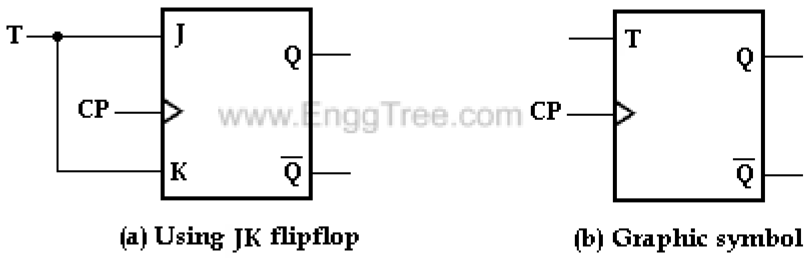
The T (*Toggle*) Flip-Flop is a modification of the JK Flip-Flop. It is obtainedfromJKFlip-Flopbyconnectingbothinputs JandKtogether,i.e.,singleinput.Regardlessofthepresentstate,theFlip-Flopcomplementsitsoutputwhentheclockpulseoccurswhileinput T=1.

TFlip-Flop

When T= 0, Q_{n+1}= Q_n, ie., the next state is the sameas the present state and nochangeoccurs.



WhenT=1,Q_{n+1}=Q_n',ie.,thenextstateisthecomplementofthepresentstate.



ThetruthtableofTFlip-Flopisgivenbelow.

T	Q_{n+1}	State
0 1	Q_n Q_n'	No ChangeT oggle

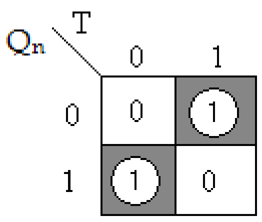
TruthtableforTFlip-Flop

CharacteristictableandCharacteristicequation:

ThecharacteristictableforTFlip-Flopisshownbelowandcharacteristicequationisderivedusing K-map.

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

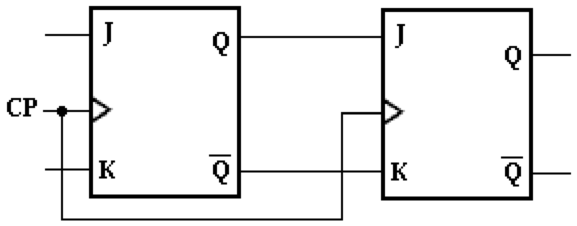
K-mapSimplification:



Characteristicequation:$Q_{n+1}=TQ_n' + T'Q_n$.

Master-SlaveJKFlip-Flop

Amaster-slaveFlip-FlopisconstructedusingtwoseparateJKFlip-Flops.The firstFlip-Flop is called the master. Itis driven by the positive edge of the clockpulse.ThesecondFlip-Flopiscalledtheslave.Itisdrivenby the negative edge oftheclockpulse.Thelogicdiagramofamaster-slaveJKFlip-Flopisshownbelow.

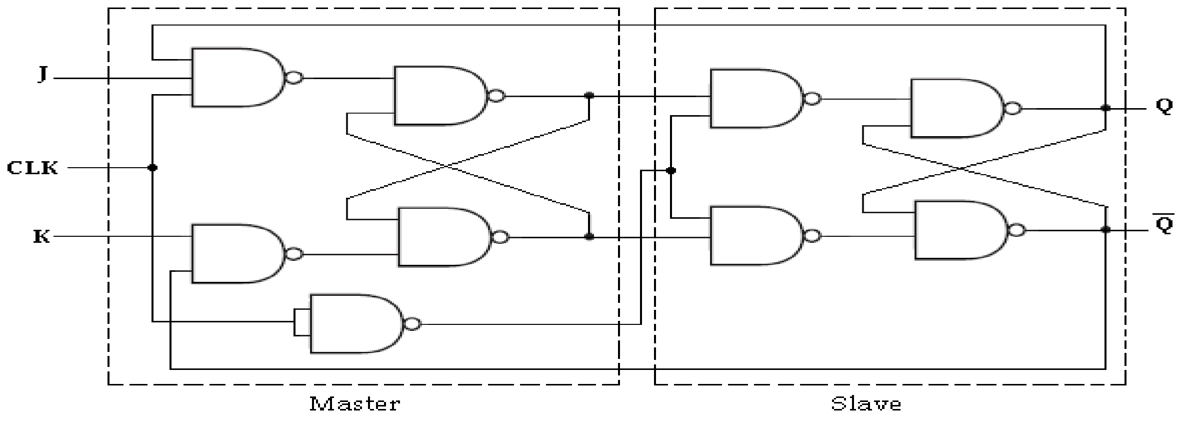


Logicdiagram

When the clock pulse has a positive edge, the master acts according to its J-K inputs, but the slave does not respond, since it requires a negative edge at theclockinput.

When the clock input has a negative edge, the slave Flip-Flop copies themaster outputs. But the master does not respond since it requires a positive edge atitsclockinput.

Theclockedmaster-slaveJ-KFlip-FlopusingNANDgatesisshownbelow.



Master-SlaveJKFlip-Flop

APPLICATION TABLE(OR)EXCITATION TABLE:

The*characteristictable*isusefulfor**analysis**andfordefiningtheoperationoftheFlip-Flop.Itspecifiesthenextstate(Q_{n+1})whentheinputsandpresentstateareknown.

The*excitationorapplicationtable*isusefulfor**design**process.Itisusedtofind theFlip-Flopinputconditions thatwill causetherequiredtransition,when thepresentstate(Q_n)andthenextstate(Q_{n+1})areknown.

SR flipflop

Present State	Inputs		Next State
Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	x
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	x

CharacteristicTable

Present State	Next State	Inputs		Inputs	
Q_n	Q_{n+1}	S	R	S	R
0	0	0	0	0	x
0	0	0	1		
0	1	1	0	1	0
1	0	0	1	0	1
1	1	0	0	x	0
1	1	1	0		

ModifiedTable

Present State	Next State	Inputs	
Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

xcitation Table

The above table presents the excitation table for SR Flip-Flop. It consists of present state (Qn), next state (Qn+1) and a column for each input to show how the required transition is achieved.
There are 4 possible transitions from present state to next state. The required Input conditions for each of the four transitions are derived from the information available in the characteristic table. The symbol ‘x’ denotes the don’t care condition, it does not matter whether the input is 0 or 1.

JKFlip-Flop:

Present State	Inputs		Next State
Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

CharacteristicTable

Present State	Next State	Inputs		Inputs	
Q_n	Q_{n+1}	J	K	J	K
0	0	0	0	0	x
0	0	0	1		
0	1	1	0	1	x
0	1	1	1		
1	0	0	1	x	1
1	0	1	1		
1	1	0	0	x	0
1	1	1	0		

ModifiedTable

Present State	Next State	Inputs	
Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Excitation Table

DFlip-Flop

Present State	Input	Next State
Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

CharacteristicTable

Present State	Next State	Input
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

ExcitationTable

TFlip-Flop

Present State	Input	Next State
Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

CharacteristicTable

Present State	Next State	Input
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

ModifiedTable

REALIZATIONOFONEFLIP-FLOPUSINGOTHERFLIP-FLOPS

It is possible to convert one Flip-Flop into another Flip-Flop with someadditional gates or simply doing some extra connection. The realization of one Flip-Flop using other Flip-Flops is implemented by the use of characteristic tables andexcitationtables.LetusseefewconversionsamongFlip-Flops.

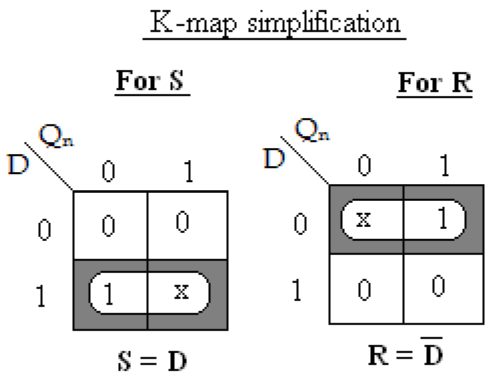
- SRFlip-FloptoDFlip-
- FlopSRFlip-FloptoJKFlip-
- FlopSRFlip-FloptoTFlip-
- FlopJKFlip-FloptoTFlip-
- FlopJKFlip-FloptoDFlip-
- FlopDFlip-FloptoTFlip-
  Flop
- TFlip-FloptoDFlip-Flop

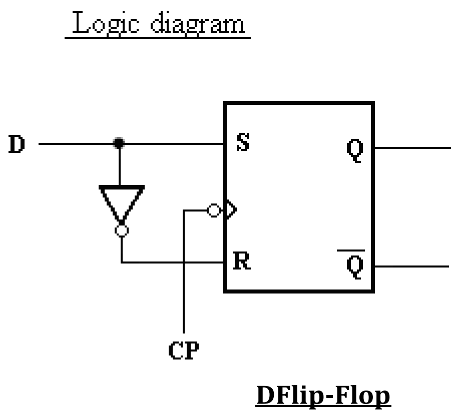
SRFlip-FloptoDFlip-Flop:

- WritethecharacteristictableforrequiredFlip-Flop(DFlip-Flop).
- WritetheexcitationtableforgivenFlip-Flop(SRFlip-Flop).
- DeterminetheexpressionforthegivenFlip-Flopinputs(SandR)byusingK-map.
- DrawtheFlip-Flopconversionlogicdiagramtoobtaintherequired Flip-Flop(D Flip-Flop)byusingtheaboveobtainedexpression.

Theexcitationtablefortheaboveconversionis

RequiredFlip-Flop(D)			**GivenFlip-Flop (SR)**	
Input	**Presentstate**	**Nextstate**	**Flip-FlopInputs**	
D	**Q_n**	**Q_{n+1}**	**S**	**R**
0	0	0	0	x
0	1	0	0	1
1	0	1	1	0
1	1	1	x	0

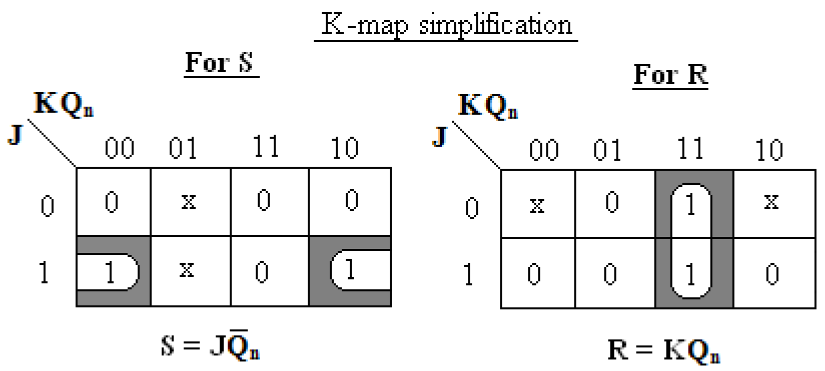


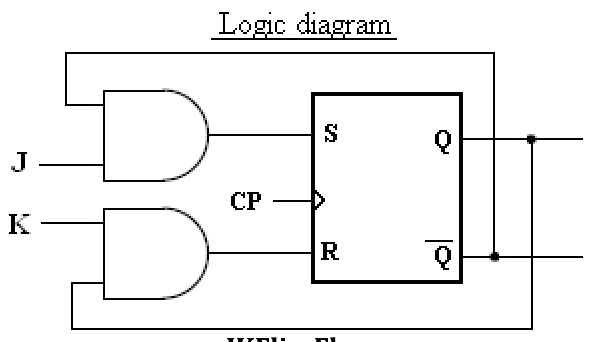


SRFlip-FloptoJKFlip-Flop

Theexcitationtablefortheaboveconversionis,

Inputs		Presentstate	Nextstate	Flip-Flop Input	
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	x
0	0	1	1	x	0
0	1	0	0	0	x
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	x	0
1	1	0	1	1	0
1	1	1	0	0	1



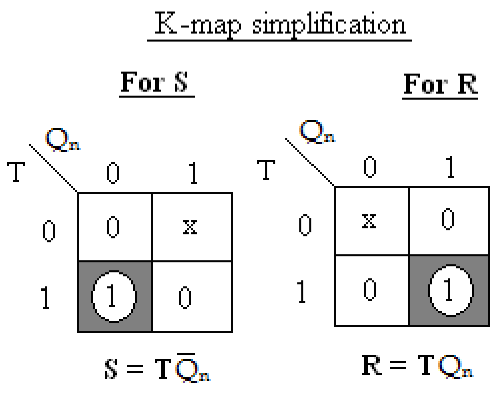


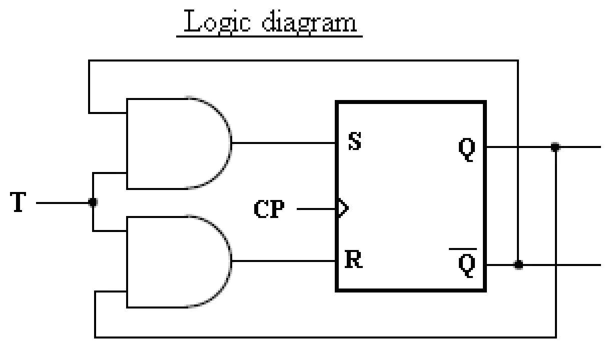
JKFlip-Flop

2.7.3 SRFlip-FloptoTFlip-Flop

Theexcitationtablefortheaboveconversionis

Input	Presentstate	Nextstate	Flip-Flop Inputs	
T	Q_n	Q_{n+1}	S	R
0	0	0	0	x
0	1	1	x	0
1	0	1	1	0
1	1	0	0	1

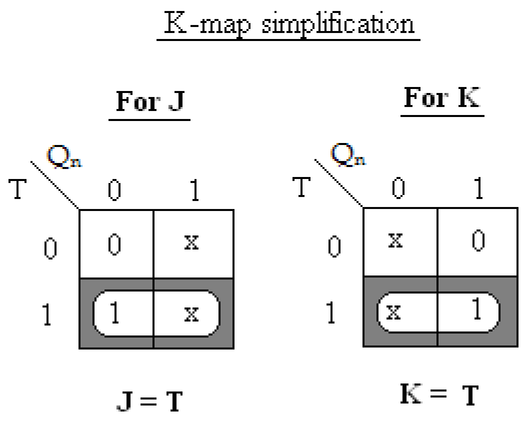


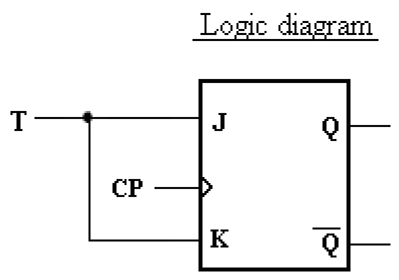


3.7.4JKFlip-FloptoTFlip-Flop

Theexcitationtablefortheaboveconversionis

Input	Presentstate	Nextstate	Flip-Flop Inputs	
T	Q_n	Q_{n+1}	J	K
0	0	0	0	x
0	1	1	x	0
1	0	1	1	x
1	1	0	x	1



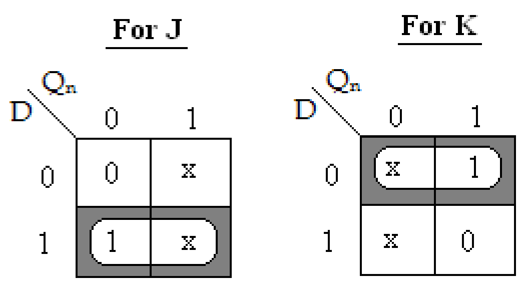


JKFlip-FloptoDFlip-Flop

Theexcitationtablefortheaboveconversionis

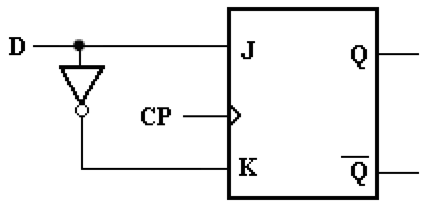
Input	Presentstate	Nextstate	Flip-Flop Inputs	
D	Q_n	Q_{n+1}	J	K
0	0	0	0	x
0	1	0	x	1
1	0	1	1	x
1	1	1	x	0

K-map simplification



J = D $K = \overline{D}$

Logic diagram

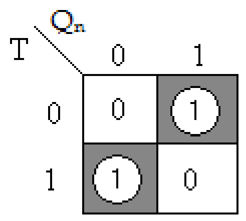


DFlip-FloptoTFlip-Flop

Theexcitationtablefortheaboveconversionis

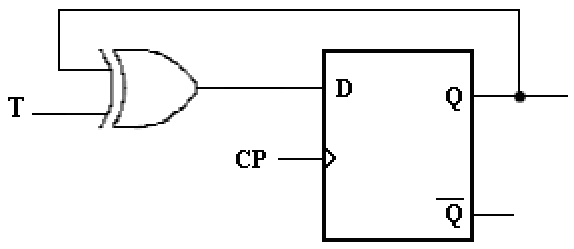
Input	Presentstate	Nextstate	Flip-Flop Input
T	Q_n	Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

K-map simplification



$$D = \overline{T}Q_n + T\overline{Q}_n$$
$$= T \oplus Q_n$$

Logic diagram

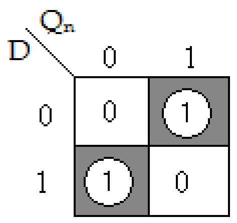


TFlip-FloptoDFlip-Flop

Theexcitationtablefortheaboveconversionis

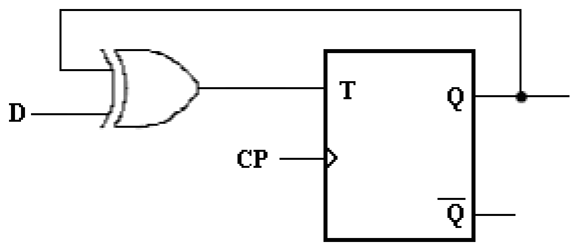
Input	Presentstate	Nextstate	Flip-Flop Input
D	Q_n	Q_{n+1}	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

K-map simplification



$$T = D\overline{Q_n} + \overline{D}Q_n$$
$$= D \oplus Q_n$$

Logic diagram



CLASSIFICATIONOFSYNCHRONOUSSEQUENTIALCIRCUIT:

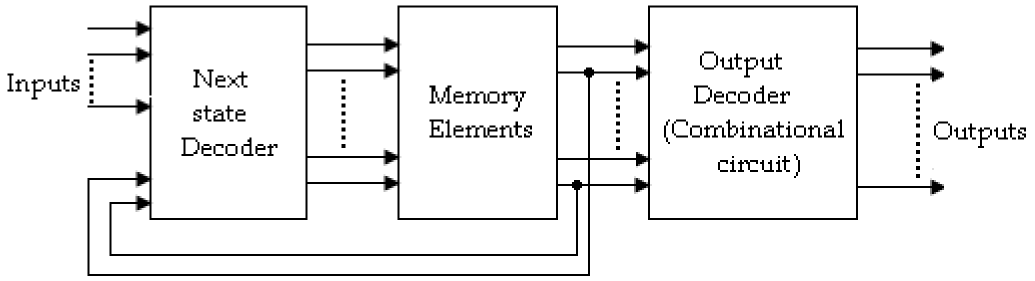
In synchronous or clocked sequential circuits, clocked Flip-Flops are used asmemory elements, which change their individual states in synchronism with theperiodic clock signal. Therefore, the change in states of Flip-Flop and change in stateof theentirecircuits occuratthetransitionoftheclocksignal.

The synchronous or clocked sequential networks are represented by twomodels.

- **Mooremodel:**TheoutputdependsonlyonthepresentstateoftheFlip-Flops.
- **Mealymodel:**TheoutputdependsonboththepresentstateoftheFlip-Flopsandontheinputs.

Mooremodel:

IntheMooremodel,theoutputsareafunctionofthepresentstateoftheFlip-Flops only. The output depends only on present state of Flip-Flops, it appears onlyafter theclock pulse isapplied, i.e.,it variesinsynchronism with the clockinput.

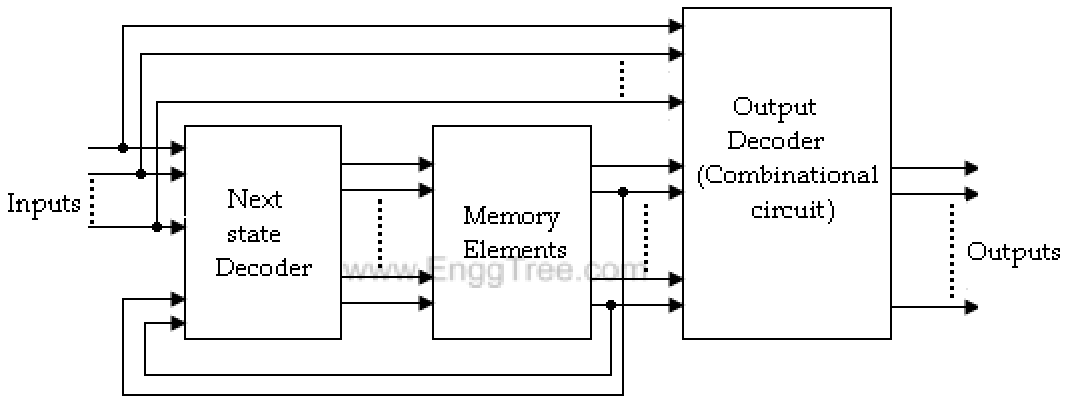


Mooremodel

Mealymodel:

IntheMealymodel,theoutputsarefunctionsofboththepresentstateoftheFlip-Flopsandinputs.

Mealymodel



DifferencebetweenMooreandMealymodel

Sl.No	Mooremodel	Mealymodel
1	Itsoutputisafunctionofpresent stateonly.	Itsoutputisafunctionofpresentstate aswellaspresentinput.
2	Inputchangesdoesnotaffectthe output.	Inputchangesmayaffecttheoutputof thecircuit.
3	Itrequiresmorenumberofstates forimplementingsamefunction.	Itrequireslessnumberofstatesfor implementingsamefunction.

ANALYSISOFSYNCHRONOUSSEQUENTIALCIRCUIT:

The behavior of a sequential circuit is determined from the inputs, outputsand the state of its Flip-Flops. The outputs and the next state are both a function ofthe inputs and the present state. The analysis of a sequential circuit consists ofobtainingatableordiagramfromthetimesequenceofinputs,outputsandinternalstates.

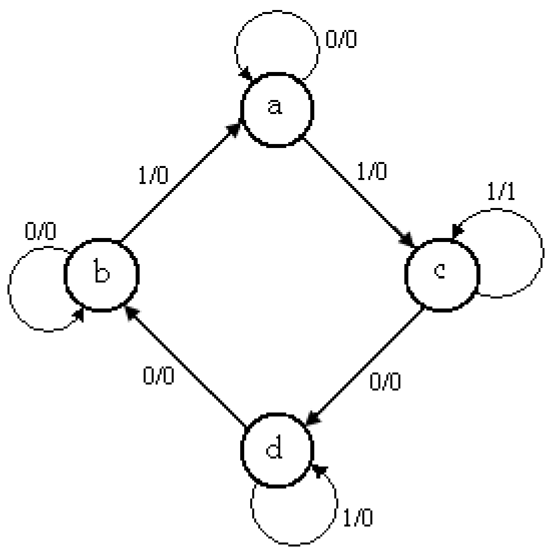
Beforegoingtoseetheanalysisanddesignexamples,wefirstunderstandthestate diagram,statetable.

StateDiagram

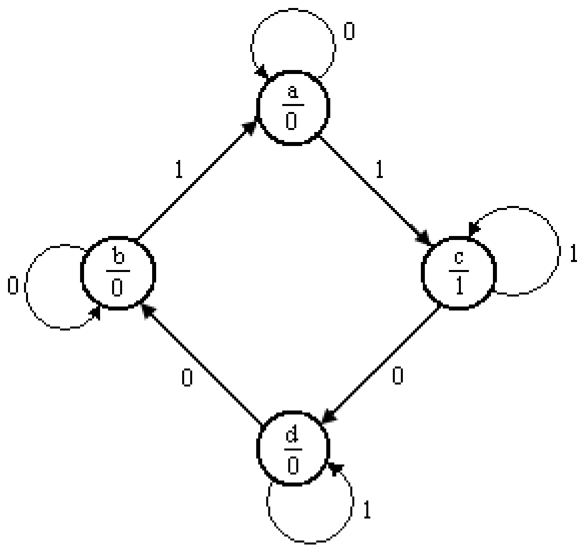
Statediagramisapictorialrepresentationofabehaviorofasequentialcircuit.

- Inthestatediagram,astateisrepresentedbyacircleandthetransitionbetweenstatesis indicatedbydirectedlinesconnectingthecircles.
- A directed line connecting a circle with circle with itself indicates that nextstateissameaspresentstate.
- Thebinarynumberinsideeach circleidentifies thestate representedbythecircle.
- Thedirectedlinesarelabeledwithtwobinarynumbersseparatedbyasymbol'/'.Theinputvaluethatcausesthestatetransitionislabeledfirstandtheoutputvalueduringthepresentstateislabeledafterthesymbol'/'.

IncaseofMoorecircuit,thedirectedlinesarelabeledwithonlyonebinarynumberrepresentingthestateoftheinputthatcausesthestatetransition.Theoutputstateisindicatedwithinthecircle,belowthepresentstatebecauseoutputstatedependsonlyonpresentstateandnotontheinput.



Statediagramfor Mealy circuit



StatediagramforMoorecircuit

StateTable

Statetablerepresents relationshipbetweeninput,outputandFlip-Flopstates.

- Itconsistsofthreesectionslabeledpresentstate,nextstateandoutput.
    - ThepresentstatedesignatesthestateofFlip-Flopsbeforetheoccurrence of a clock pulse, and the output section gives the values oftheoutputvariablesduringthepresentstate.
    - Both the next state and output sections have two columns representingtwopossibleinputconditions:X=0andX=1.

Presentstate	**Nextstate**		**Output**	
	X=0	**X=1**	**X=0**	**X=1**
AB	AB	AB	Y	Y
a	a	c	0	0
b	b	a	0	0
c	d	c	0	1
d	b	d	0	0

- IncaseofMoorecircuit,theoutputsectionhasonlyonecolumnsinceoutputdoesnot dependoninput.

Presentstate	**Nextstate**		**Output**
	X=0	**X=1**	**Y**
AB	AB	AB	
a	a	c	0
b	b	a	0
c	d	c	1
d	b	d	0

2.9.3 StateEquation

ItisanalgebraicexpressionthatspecifiestheconditionforaFlip-Flopstatetransition.
The Flip-Flopsmay beof any typeandthe logicdiagrammay ormaynotincludecombinationalcircuitgates.

ANALYSISPROCEDURE

Thesynchronoussequentialcircuitanalysisissummarizesasgivenbelow:

1. AssignastatevariabletoeachFlip-Flopinthesynchronoussequentialcircuit.
2. WritetheexcitationinputfunctionsforeachFlip-FlopandalsowritetheMoore/Mealyoutputequations.
3. SubstitutetheexcitationinputfunctionsintothebistableequationsfortheFlip-Flopstoobtainthenextstateoutputequations.
4. Obtainthestatetableandreducedformofthestatetable.
5. Drawthestatediagrambyusingthesecondformofthestatetable.

Analysis ofMealy Model

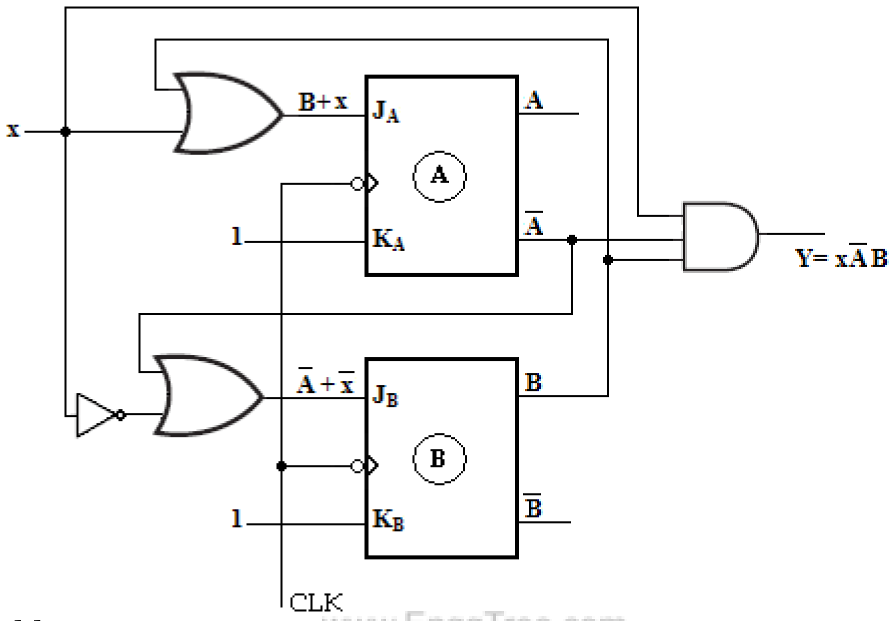
1. A sequentialcircuithas two JKFlip-Flops Aand B,oneinput(x) andoneoutput(y).theFlip-Flopinputfunctionsare,

   J_A=B+x **J_B=A'+x'**
   K_A=1 **K_B=1**

   andthecircuitoutputfunction,**Y=xA'B**.

a) DrawthelogicdiagramoftheMealycircuit,
b) Tabulatethestatetable,
c) Drawthestatediagram.

Soln:



Statetable:

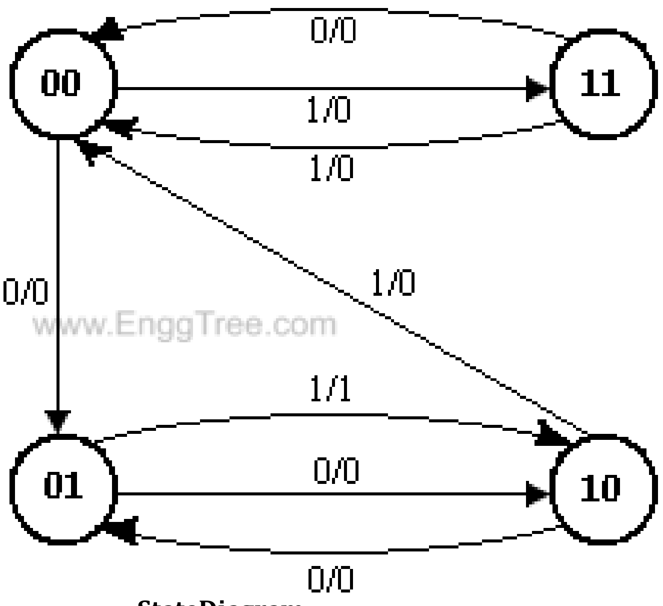
Toobtainthenext-statevaluesofasequentialcircuitwithJKFlip-Flops,usetheJKFlip-Flopcharacteristicstable.

Presentstate		Input	Flip-FlopInputs				Nextstate		Output
A	B	x	J_A=B+x	K_A=1	J_B=A'+x'	K_B=1	A(t+1)	B(t+1)	Y=xA'B
0	0	0	0	1	1	1	0	1	0
0	0	1	1	1	1	1	1	1	0
0	1	0	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0	1	0
1	0	1	1	1	0	1	0	0	0
1	1	0	1	1	1	1	0	0	0
1	1	1	1	1	0	1	0	0	0

Presentstate		Nextstate				Output	
		x=0		x=1		x=0	x=1
A	B	A	B	A	B	y	y
0	0	0	1	1	1	0	0
0	1	1	0	1	0	0	1
1	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0

Second form ofstatetable

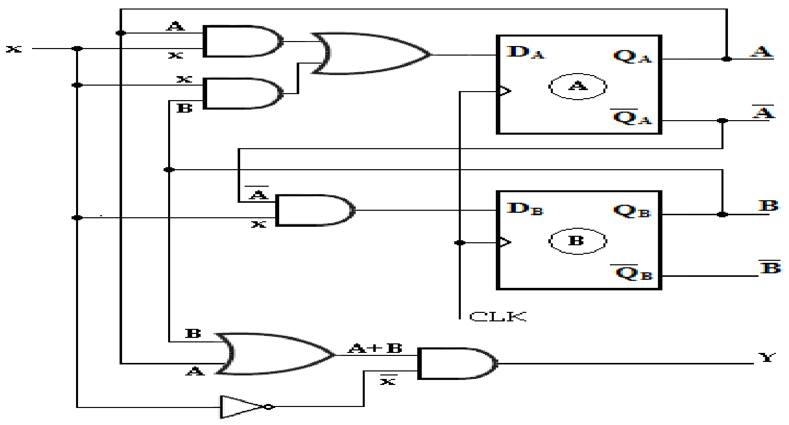
State Diagram:



StateDiagram

2. Asequentialcircuitwithtwo'D'Flip-FlopsAandB,oneinput(x)andoneoutput(y).theFlip-Flopinputfunctionsare:
   D_A=Ax+Bx
   D_B=A'x andthecircuitoutputfunctionis,
   Y=(A+ B)x'.
   (a) Drawthe logicdiagram of the circuit,
   (b) Tabulatethestatetable,
   (c) Drawthestatediagram.

Soln:



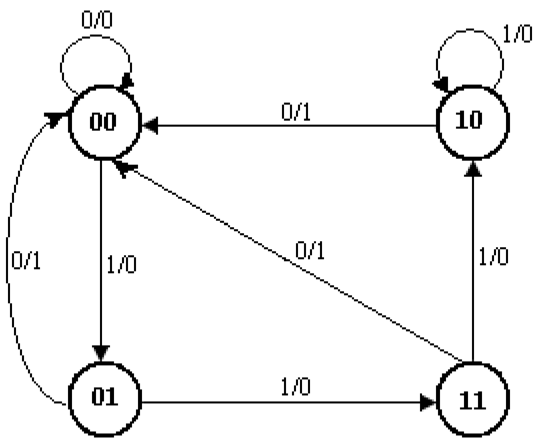
State Table:

Presentstate		Input	Flip-FlopInputs		Nextstate		Output
A	B	x	D_A= Ax+Bx	D_B=A'x	A(t+1)	B(t+1)	Y=(A+B)x'
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	1
0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	1
1	1	1	1	0	1	0	0

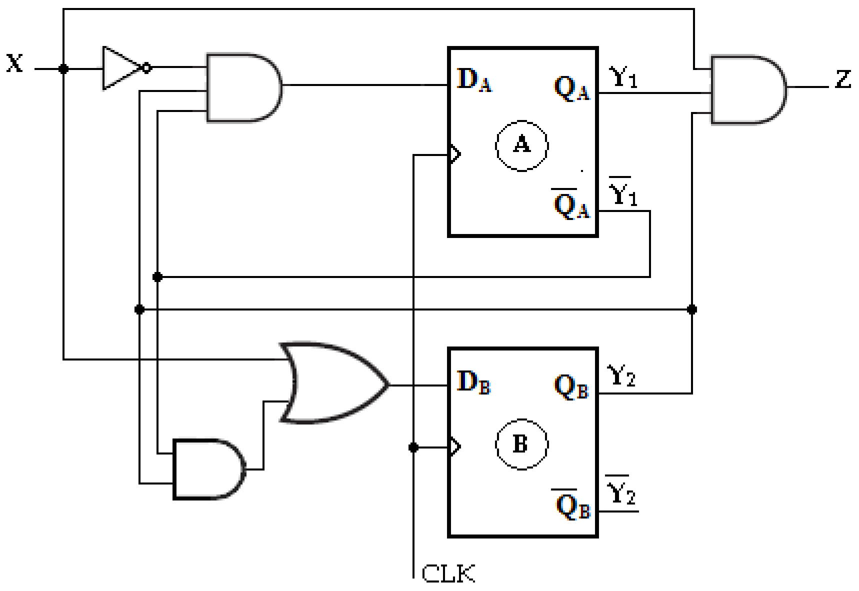
Presentstate		Nextstate				Output	
		x=0		x=1		x=0	x=1
A	B	A	B	A	B	Y	Y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

Secondform ofstatetable

StateDiagram:



3. AnalyzethesynchronousMealymachineandobtainitsstatediagram.



Soln:
The given synchronous Mealy machine consists of two D Flip-Flops, one inputs andoneoutput.
TheFlip-Flopinputfunctionsare,

D_A=Y_1'Y_2X'D
$_B$=X+Y_1'Y_2

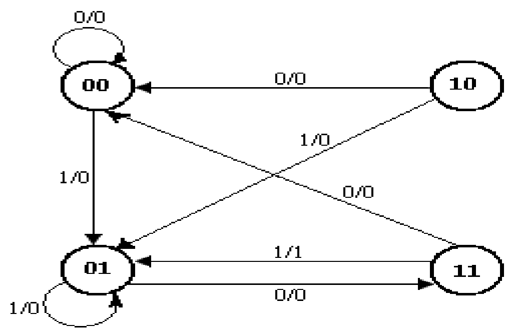
Thecircuitoutputfunctionis,**Z=Y_1Y_2X**

State Table:

Presentstate		Input	Flip-FlopInputs		Nextstate		Output
Y_1	Y_2	X	D_A=Y_1'Y_2X'	D_B=X+Y_1'Y_2	Y_1(t+1)	Y_2(t+1)	Z=Y_1Y_2X
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
1	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	0
1	1	1	0	1	0	1	1

Presentstate		Nextstate				Output	
		X=0		X=1		X=0	X=1
Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Z	Z
0	0	0	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	0
1	1	0	0	0	1	0	1

Secondform ofstatetable



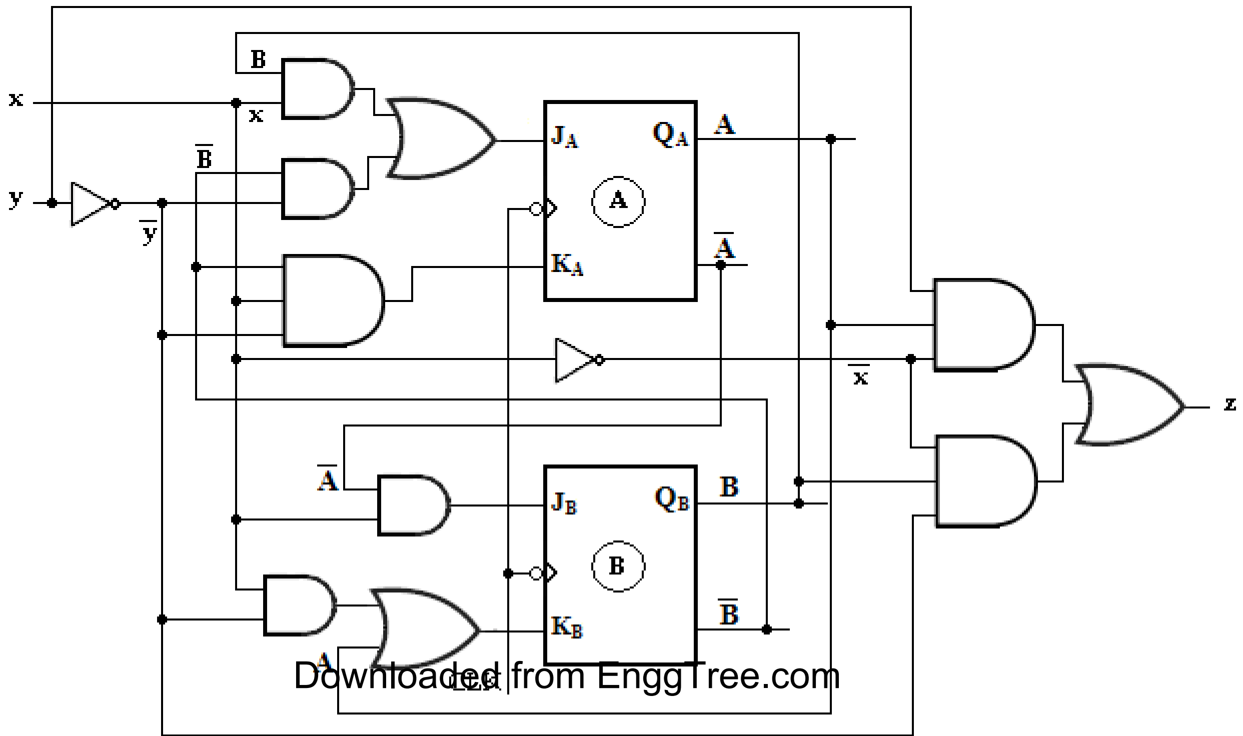
StateDiagram:

4. AsequentialcircuithastwoJKFlop-FlopsAandB,twoinputsxandyandoneoutput z.The Flip-Flop inputequation andcircuit output equationsare

J_A =Bx+B'y' **K_A=B'xy'**
J_B =A'x **K_B=A+xy'**
z=Ax'y'+Bx'y'

(a) Drawthelogicdiagramofthecircuit
(b) Tabulatethestatetable.
(c) Derivethestateequation.

Statediagram:

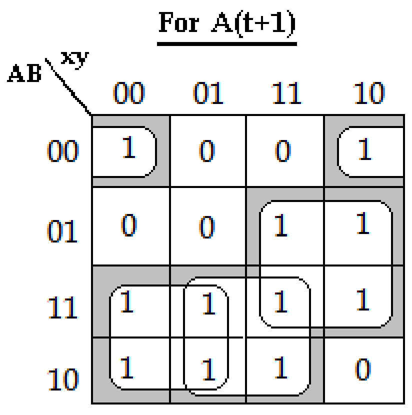


Statetable:

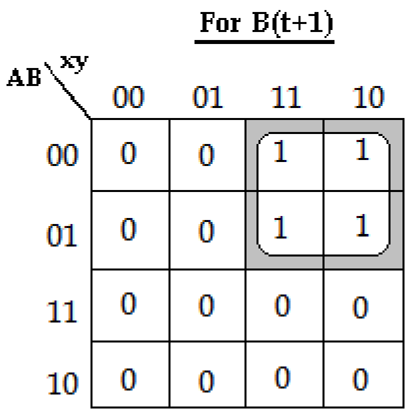
Toobtainthenext-statevaluesofasequentialcircuitwithJKFlip-Flop,usetheJKFlip-Flopcharacteristictable,

Present state		Input		Flip-FlopInputs				Nextstate		Output
A	B	x	y	J_A= Bx+B'y'	K_A= B'xy'	J_B= A'x	K_B= A+xy'	A(t+1)	B(t+1)	z
0	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	1	1	1	1	1	0
0	0	1	1	0	0	1	0	0	1	0
0	1	0	0	0	0	0	0	0	0	1
0	1	0	1	0	0	0	0	0	0	0
0	1	1	0	1	0	1	1	1	1	0
0	1	1	1	1	0	1	0	1	1	0
1	0	0	0	1	0	0	1	1	0	1
1	0	0	1	0	0	0	1	1	0	0
1	0	1	0	1	1	0	1	0	0	0
1	0	1	1	0	0	0	1	1	0	0
1	1	0	0	0	0	0	1	1	0	1
1	1	0	1	0	0	0	1	1	0	0
1	1	1	0	1	0	0	1	1	0	0
1	1	1	1	1	0	0	1	1	0	0

State Equation:



A (t+1) = Ax' + Ay + Bx + A'B'y'



B (t+1) = A'x

5. AsequentialcircuithastwoJKFlip-FlopAandB.theFlip-Flopinputfunctionsare:

J_A=B J_B=x

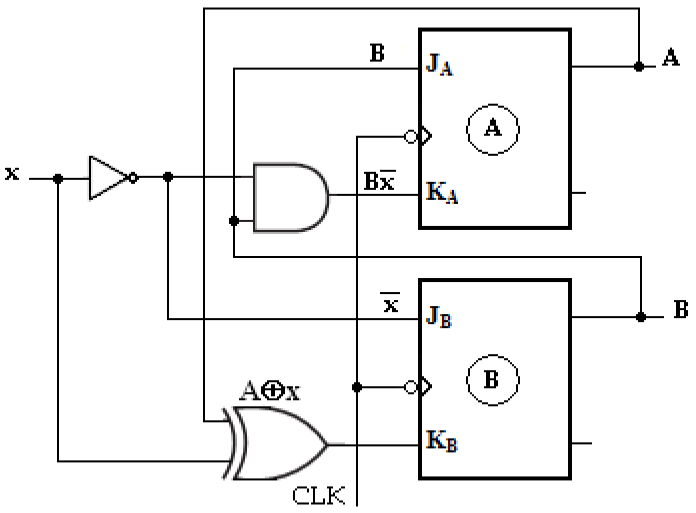
K_A=Bx' **K_B=A⊕x.**

(a) Drawthe logicdiagram of the circuit,

(b) Tabulatethestatetable,

(c) Drawthestatediagram.

Logic diagram:



Theoutputfunctionisnotgivenintheproblem.Theoutputofthe Flip-Flopsmaybeconsideredastheoutputofthecircuit.

Statetable:

Toobtainthenext-statevaluesofasequentialcircuitwithJKFlip-Flop,usetheJKFlip-Flopcharacteristictable.

Presentstate		Input	Flip-FlopInputs				Nextstate	
A	B	x	J_A=B	K_A=Bx'	J_B=x'	K_B=A x	A(t+1)	B(t+1)
0	0	0	0	0	1	0	0	1
0	0	1	0	0	0	1	0	0
0	1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	1	0
1	0	0	0	0	1	1	1	1
1	0	1	0	0	0	0	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	0	0	0	1	1

Presentstate		Nextstate			
		X=0		X=1	
A	B	A	B	A	B
0	0	0	1	0	0
0	1	1	1	1	0
1	0	1	1	1	0
1	1	0	0	1	1

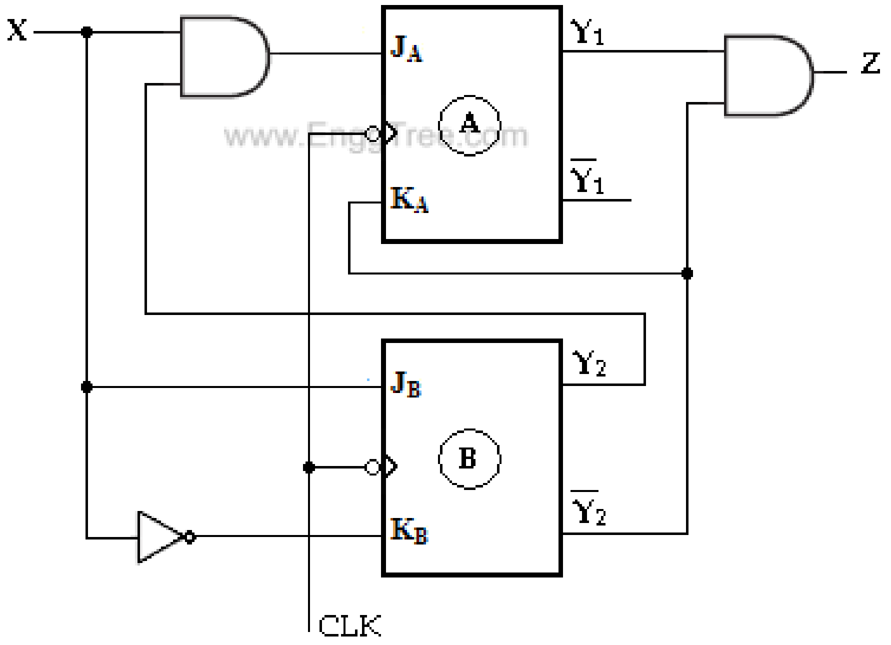
Secondformofstatetable

StateDiagram:

assignedvariable $Y_1$and Y_2forthetwoJKFlip-Flops,we canwrite thefourexcitationinputequationsandtheMooreoutputequat

$J_A=Y_2X$; **$K_A=Y_2'$** ionasfollows

$J_B=X$; **$K_B=X'$** and outputfunction,**$Z=Y_1Y_2'$** :



AnalysisofMooreModel

6. Analyzethe synchronousMoorecircuitandobtainitsstatediagram.

Soln:

Using the

Secondformofstatetable

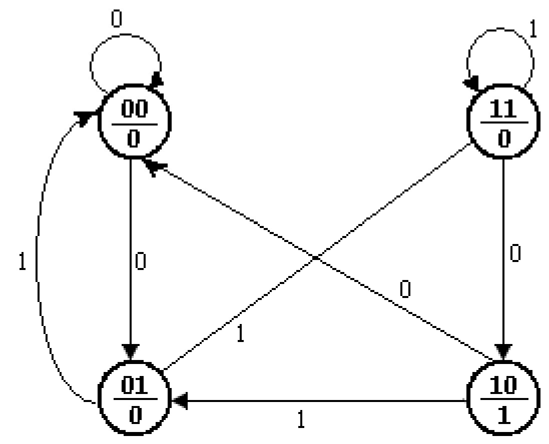
Statetable:

Presentstate		Input	Flip-FlopInputs				Nextstate		Output
Y_1	Y_2	X	$J_A=Y_2X$	$K_A=Y_2'$	$J_B=X$	$K_B=X'$	$Y_1(t+1)$	$Y_2(t+1)$	$Z=Y_1Y_2'$
0	0	0	0	1	0	1	0	0	0
0	0	1	0	1	1	0	0	1	0
0	1	0	0	0	0	1	0	0	0
0	1	1	1	0	1	0	1	1	0
1	0	0	0	1	0	1	0	0	1
1	0	1	0	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0	0
1	1	1	1	0	1	0	1	1	0

Presentstate		Nextstate				Output
		X=0		X=1		Y
Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	
0	0	0	0	0	1	0
0	1	0	0	1	1	0
1	0	0	0	0	1	1
1	1	1	0	1	1	0

StateDiagram:

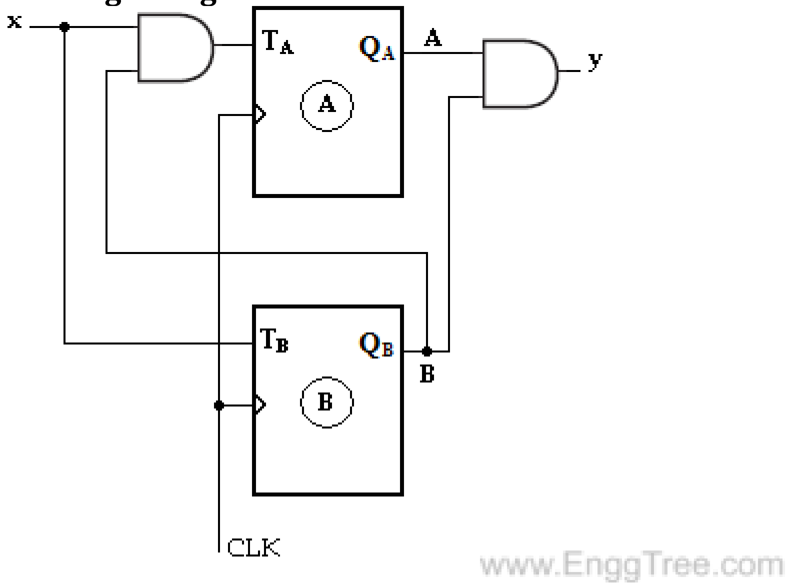
Here the output depends on the present state only and is independent of theinput.Thetwovaluesinsideeachcircleseparatedbyaslashareforthepresentstatean doutput.



7. Asequentialcircuithas twoTFlip-FlopAandB.TheFlip-Flopinputfunctionsare:
   T_A=Bx **T_B=x**
   y=AB
   (a) Drawthe logicdiagram of the circuit,
   (b) Tabulatethestatetable,
   (c) Drawthestatediagram.

Soln:

Logic diagram:



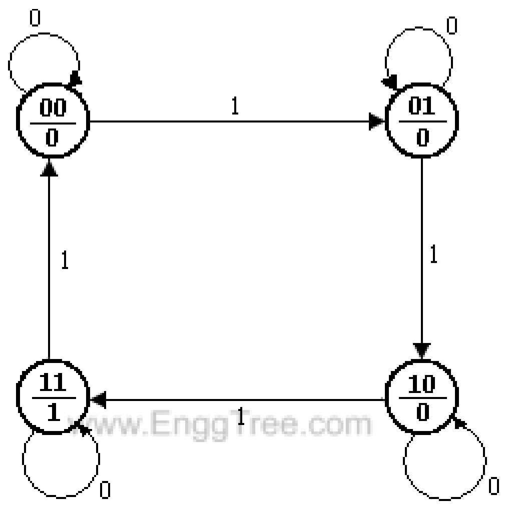
Statetable

Presentstate		Input	Flip-FlopInputs		Nextstate		Output
A	**B**	**x**	**T_A=Bx**	**T_B=x**	**A(t+1)**	**B(t+1)**	**y=AB**
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	0	1	0
0	1	1	1	1	1	0	0
1	0	0	0	0	1	0	0
1	0	1	0	1	1	1	0
1	1	0	0	0	1	1	1
1	1	1	1	1	0	0	1

Presentstate		Nextstate				Output	
		x=0		x=1		x=0	x=1
A	**B**	**A**	**B**	**A**	**B**	**y**	**y**
0	0	0	0	0	1	0	0
0	1	0	1	1	0	0	0
1	0	1	0	1	1	0	0
1	1	1	1	0	0	1	1

Secondformofstatetable

StateDiagram:



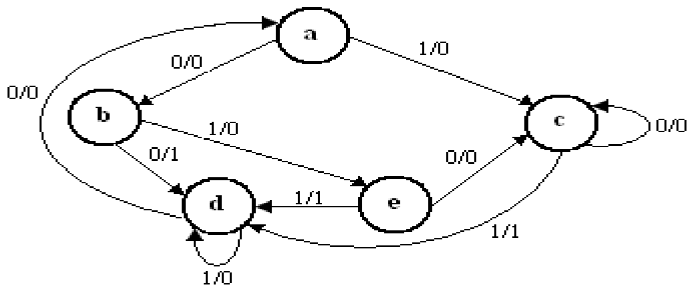
STATEREDUCTION/MINIMIZATION

The state reduction is used to avoid the redundant states in the sequentialcircuits. The reduction in redundant states reduces the number of required Flip-Flopsandlogicgates,reducingthecostofthefinalcircuit.

The two states are said to be redundant or equivalent, if every possible set ofinputs generateexactlysameoutputand samenextstate.When twostates areequivalent,oneofthemcanberemovedwithoutalteringtheinput-outputrelationship.

Since‘n’Flip-Flopsproduced2^nstate,areductioninthenumberofstatesmayresultinareductioninthenumberofFlip-Flops.

Theneedforstatereductionorstateminimizationisexplainedwithoneexample.



Statediagram

Step1:Determinethestatetableforgivenstatediagram

Presentstate	**Nextstate**		**Output**	
	X=0	**X=1**	**X=0**	**X=1**
a	b	c	0	0
b	d	e	1	0
c	c	d	0	1
d	a	d	0	0
e	c	d	0	1

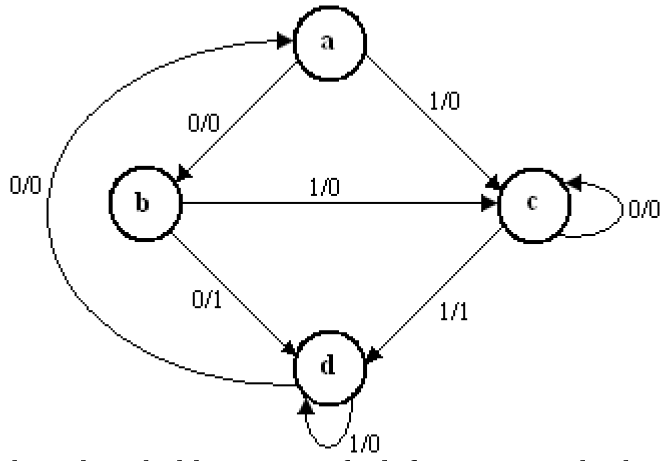
Statetable

Step2:Findequivalentstates

Fromtheabovestatetable**c**and**e**generateexactlysamenextstateandsameoutputforeverypossiblesetofinputs.Thestate**c**and**e**gotonextstates**c**and**d**andhaveoutputs0and1forx=0andx=1respectively.Thereforestate**e**canberemovedand replacedby**c**.Thefinalreducedstatetableisshownbelow

.Presentstate	**Nextstate**		**Output**	
	X=0	**X=1**	**X=0**	**X=1**
a	b	c	0	0
b	d	c	1	0
c	c	d	0	1
d	a	d	0	0

Reducedstatetable



Thestatediagramforthereducedtableconsistsofonlyfourstatesandisshownbelow.

Reducedstatediagram

1. Reducethenumberofstatesinthefollowingstatetableandtabulatethereducedstatetable.

Presentstate	Nextstate		Output	
	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

Soln:

Fromtheabovestatetable**e**and**g**generateexactlysamenextstateandsame outputforeverypossiblesetofinputs.Thestate**e**and**g**gotonextstates**a**and**f**andhaveoutputs0and1forx=0andx=1respectively.Thereforestate**g**canberemovedandreplacedby **e**.

Thereducedstatetable-1isshownbelow.

Presentstate	Nextstate		Output	
	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1

e	a	f	0	1
f	**e**	f	0	1

Reducedstatetable-1

Nowstatesdandfareequivalent.Bothstatesgotothesamenextstate(e,f)andhav esameoutput(0,1).Thereforeonestatecanberemoved;**f**isreplacedby**d.**Thefinalreduc edstatetable-2isshownbelow.

Presentstate	Nextstate		Output	
	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	**d**	0	0
d	e	**d**	0	1
e	a	**d**	0	1

Reducedstatetable-2

Thus7statesarereducedinto5states.

2. Determineaminimalstatetableequivalentfurnishedbelow

Presentstate	Nextstate	
	X=0	X=1
1	1, 0	1, 0
2	1, 1	6, 1
3	4, 0	5, 0
4	1, 1	7, 0
5	2, 0	3, 0
6	4, 0	5, 0
7	2, 0	3, 0

Soln:

Presentstate	Nextstate		Output	
	X=0	X=1	X=0	X=1
1	1	1	0	0
2	1	6	1	1
3	4	5	0	0
4	1	7	1	0
5	2	3	0	0
6	4	5	0	0
7	2	3	0	0

Fromtheabovestatetable,**5**and**7**generateexactlysamenextstateandsameoutputfor everypossiblesetofinputs.Thestate**5**and**7**gotonextstates**2**and**3**andhaveoutputs0and

0forx=0andx=1respectively.Thereforestate**7**canberemovedandreplacedby**5.**

Similarly,**3**and**6**generateexactlysamenextstateandsameoutputforeverypossiblesetofinputs.Thestate**3**and**6**gotonextstates**4**and**5**andhaveoutputs0and 0 for x=0 and x=1 respectively. Therefore state **6** can be removed and replacedby**3**.

Thefinalreducedstatetableisshownbelow.

Presentstate	**Nextstate**		**Output**	
	X=0	**X=1**	**X=0**	**X=1**
1	1	1	0	0
2	1	3	1	1
3	4	5	0	0
4	1	5	1	0
5	2	3	0	0

Reducedstatetable

Thus7statesarereducedinto5states.

3. Minimizethefollowing statetable.

Presentstate	**Nextstate**	
	X=0	**X=1**
A	D,0	C,1
B	E,1	A,1
C	H, 1	D,1
D	D,0	C,1
E	B,0	G,1
F	H, 1	D,1
G	A,0	F,1
H	C,0	A,1
I	G,1	H,1

Soln:

Presentstate	**Nextstate**		**Output**	
	X=0	**X=1**	**X=0**	**X=1**
A	D	C	0	1
B	E	A	1	1
C	H	D	1	1
D	D	C	0	1
E	B	G	0	1
F	H	D	1	1
G	A	F	0	1
H	C	A	0	1
I	G	H	1	1

From the above state table, **A** and **D** generate exactly same next state andsame output for every possible set of inputs. The state **A** and **D** go to next states **D**and **C** and have outputs 0 and 1 for x=0 and x=1 respectively. Therefore state **D** canbe removed and replaced by **A**.Similarly, **C** and **F** generate exactly same next stateand same output for every possible set of inputs. The state **C** and **F** go to next states**H** and **D** and have outputs 1 and 1 for x=0 and x=1 respectively. Therefore state **F**canberemovedandreplacedby**C**.

Thereducedstatetable-1isshownbelow.

Presentstate	**Nextstate**		**Output**	
	X=0	**X=1**	**X=0**	**X=1**
A	A	C	0	1
B	E	A	1	1
C	H	A	1	1
E	B	G	0	1
G	A	C	0	1
H	C	A	0	1
I	G	H	1	1

Reducedstatetable-1

From the above reduced state table-1, **A** and **G** generate exactly same nextstate and same output for every possible set of inputs. The state **A** and **G** go to nextstates **A** and **C** and have outputs 0 and 1 for x=0 and x=1 respectively. Thereforestate **G** can be removed and replaced by **A**. The final reduced state table-2 is shownbelow.

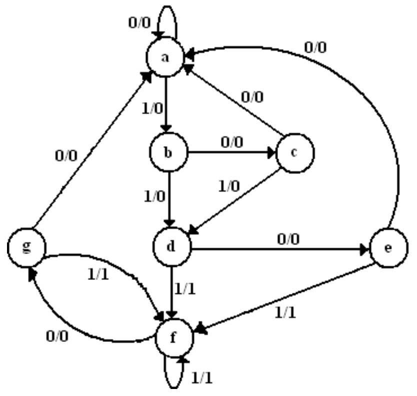
Presentstate	**Nextstate**		**Output**	
	X=0	**X=1**	**X=0**	**X=1**
A	A	C	0	1
B	E	A	1	1
C	H	A	1	1
E	B	A	0	1
H	C	A	0	1
I	A	H	1	1

Reducedstatetable-2

Thus9statesarereducedinto6states.

4. Reducethefollowingstatediagram.

Soln:



Presentstate	**Nextstate**	**Output**

	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

Statetable

Fromtheabovestatetable**e**and**g**generateexactlysamenextstateandsameoutput foreverypossiblesetofinputs.Thestate**e**and**g**gotonextstates**a**and**f**andhaveoutputs0a nd1forx=0andx=1respectively.Thereforestate**g**canberemovedand replacedby**e**.Thereducedstatetable-1is shownbelow.

Presentstate	Nextstate		Output	
	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	**e**	f	0	1

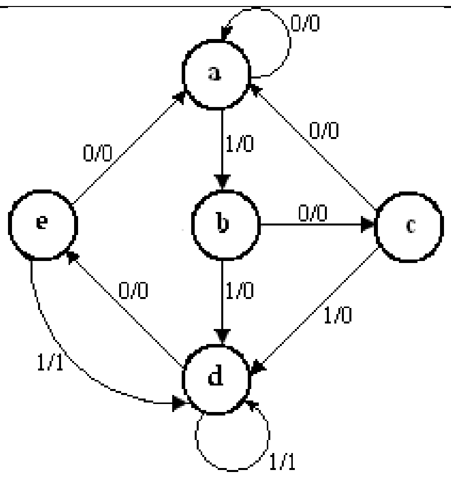
Reducedstatetable-1

Nowstatesdandfareequivalent.Bothstatesgotothesamenextstate(e,f)andhav esameoutput(0,1).Thereforeonestatecanberemoved;**f**isreplacedby**d.**Thefinalreduc edstatetable-2isshownbelow

Presentstate	Nextstate		Output	
	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	**d**	0	0
d	e	**d**	0	1
e	a	**d**	0	1

Reducedstatetable-2

Thus7statesarereducedinto5states.
Thestatediagramforthereducedstatetable-2is,



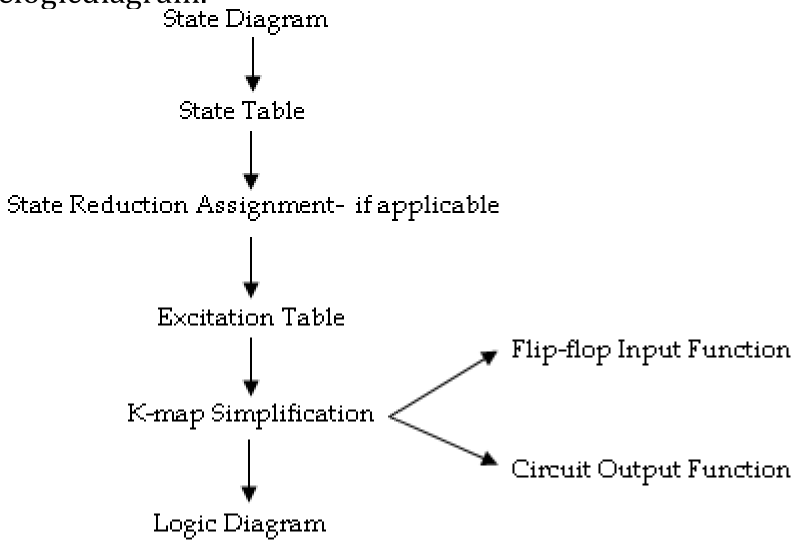
Reducedstatediagram

DESIGNOFSYNCHRONOUSSEQUENTIALCIRCUITS:

A synchronous sequential circuit is made up of number ofFlip-Flops andcombinational gates. The design of circuit consists of choosing the Flip-Flops andthenfindingacombinationalgatestructuretogetherwiththeFlip-Flops.Thenumber of Flip-Flops is determined from the number of states needed in the circuit.Thecombinationalcircuitis derivedfromthestatetable.

Designprocedure:

1. Thegivenproblemisdeterminedwithastatediagram.
2. Fromthestatediagram,obtainthestatetable.
3. Thenumberofstatesmaybereducedbystatereductionmethods(ifappl icable).
4. Assignbinaryvaluestoeachstate(BinaryAssignment)ifthestatetablecont ainslettersymbols.
5. DeterminethenumberofFlip-Flops and assignalettersymbol(A,B,C,...)toeach.
6. ChoosethetypeofFlip-Flop(SR,JK,D,T)tobeused.
7. Fromthestatetable,circuitexcitationandoutputtables.
8. UsingK-maporanyothersimplificationmethod,derivethecircuitoutputfunctionsan dtheFlip-Flopinputfunctions.
9. Drawthelogicdiagram.



The type of Flip-Flop to be used may be included in the design specificationsormaydependwhatisavailabletothedesigner.Manydigitalsystemsareco nstructed with JK Flip-Flops because they are the most versatile available. Theselectionofinputsisgivenasfollows.

Flip-Flop	Application
JK D T	GeneralApplicationsApplications requiringtransferofdata (Ex: Shift Registers)Applicationinvolving complementation (Ex:BinaryCounters)

ExcitationTables:

Beforegoingtothedesignexamplesfortheclockedsynchronoussequentialcircuitswere viseFlip-Flopexcitationtables.

Present State	Next State	Inputs	
Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

ExcitationtableforSRFlip-Flop

Present State	Next State	Inputs	
Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

ExcitationtableforJKFlip-Flop

Present State	Next State	Input
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Excitationtable forTFlip-Flop

Present State	Next State	Input
Q_n	Q_{n+1}	D

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

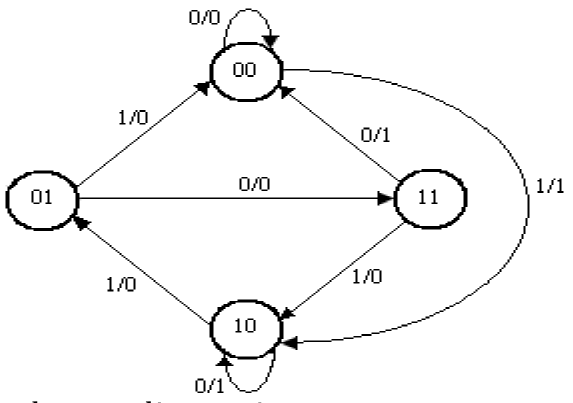
<u>**ExcitationtableforDFlip-Flop**</u>

Problems

1. A sequential circuit has one input and one output. The state diagram is shownbelow. Design the sequential circuit with a) D-Flip-Flops, b) T Flip-Flops, c) RSFlip-Flopsandd)JKFlip-Flops.

<u>**Solution:**</u>

State Table:



Thestatetableforthestatediagramis,

Presentstate		Nextstate		Output	
		X=0	X=1	X=0	X=1
A	B	AB	AB	Y	Y
0	0	00	10	0	1
0	1	11	00	0	0
1	0	10	01	1	0
1	1	00	10	1	0

Statereduction:

Asseenfromthestatetablethereisnoequivalentstates.Therefore,noreductioninthestatediagram.

The state table shows that circuit goes through four states, therefore werequire2Flip-Flops(numberofstates=2^m,wherem=numberofFlip-Flops).Sincetwo Flip-Flopsare requiredfirst isdenoted asA andsecondisdenotedasB.

i) <u>DesignusingDFlip-Flops</u>:

Excitationtable:

UsingtheexcitationtableforTFlip-Flop,wecandeterminetheexcitationtableforthe givencircuitas,

PresentState	NextState	Input
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

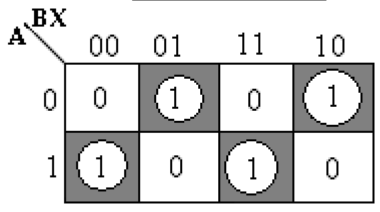
ExcitationtableforDFlip-Flop

Presentstate		Input	Nextstate		Flip-Flop Inputs		Output
A	B	X	A	B	D_A	D_B	Y
0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	1
0	1	0	1	1	1	1	0
0	1	1	0	0	0	0	0
1	0	0	1	0	1	0	1
1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	1
1	1	1	1	0	1	0	0

Circuitexcitationtable

K-mapSimplification:

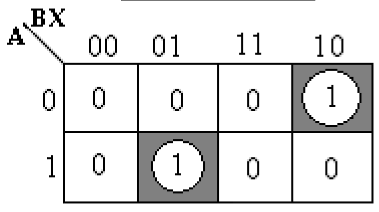
For Flip-flop A



$D_A = A'B'X + A'BX' + ABX + AB'X'$

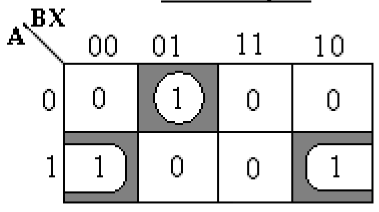
$= A \oplus (B \oplus x)$

For Flip-flop B



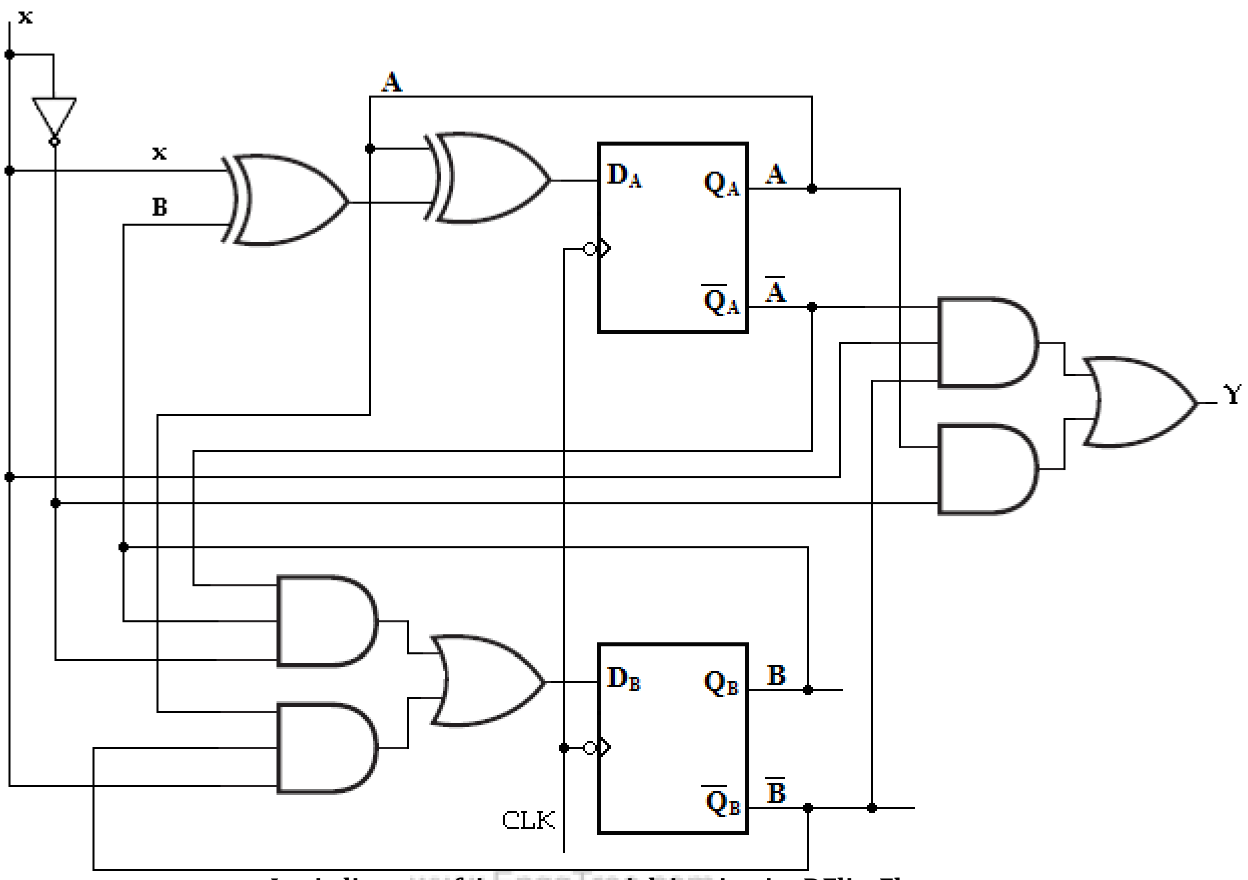
$D_B = A'BX' + AB'X$

For Output



$Y = A'B'X + AX'$

WiththeseFlip-Flopinputfunctionsandcircuitoutputfunctionwecandrawthelogicdiagramasfollows.



Logicdiagramofgivensequentialcircuitusing DFlip-Flop

ii) **DesignusingTFlip-Flops:**

UsingtheexcitationtableforTFlip-Flop,wecandeterminetheexcitationtableforthegivencircuitas,

PresentState	NextState	Input
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

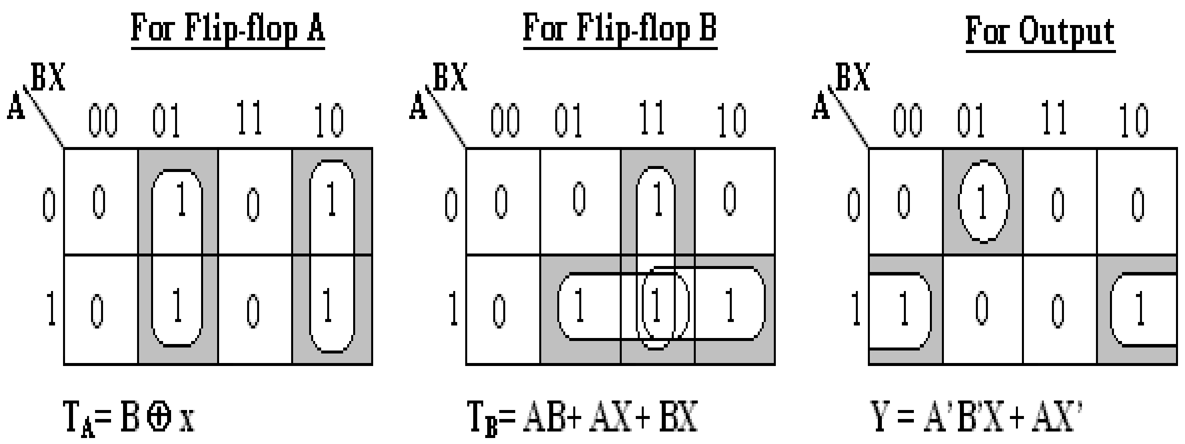
Excitation tablefor TFlip-Flop

Presentstate		Input	Nextstate		Flip-Flop Inputs		Output
A	B	X	A	B	T_A	T_B	Y
0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	1
0	1	0	1	1	1	0	0
0	1	1	0	0	0	1	0

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

Circuitexcitationtable

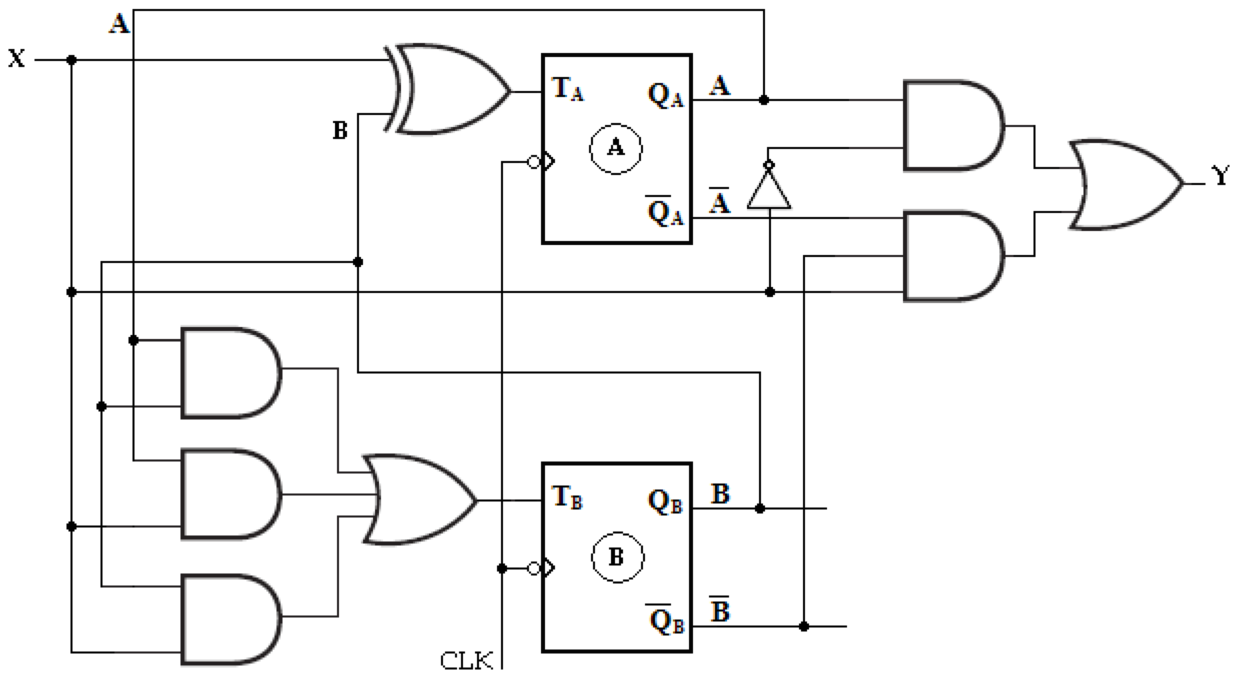
K-mapSimplification:



Therefore,inputfunctionsfor,

T_A= B ⊕x
andT_B=AB+AX+B
X
Circuitoutputfunction,**Y=XA’B’+X’A**

WiththeseFlip-
Flopinputfunctionsandcircuitoutputfunctionwecandrawthelogicdiagramasfollows.



Logicdiagramofgivensequentialcircuitusing TFlip-Flop

iii) DesignusingSRFlip-Flops:

UsingtheexcitationtableforRSFlip-

Flop,wecandeterminetheexcitationtableforthegivencircuitas,

PresentState	NextState	Inputs	
Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

ExcitationtableforSRFlip-Flop

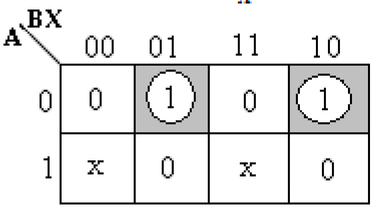
Present state		Input	Nextstate		Flip-FlopInputs				Output
A	B	X	A	B	S_A	R_A	S_B	R_B	Y
0	0	0	0	0	0	x	0	x	0
0	0	1	1	0	1	0	0	x	1
0	1	0	1	1	1	0	x	0	0
0	1	1	0	0	0	x	0	1	0
1	0	0	1	0	x	0	0	x	1
1	0	1	0	1	0	1	1	0	0
1	1	0	0	0	0	1	0	1	1
1	1	1	1	0	x	0	0	1	0

Circuitexcitationtable

K-mapSimplification:

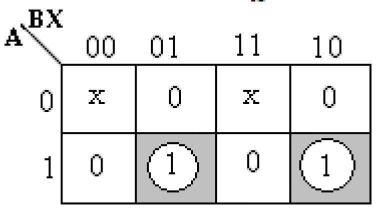
For Flip-flop A

For S_A



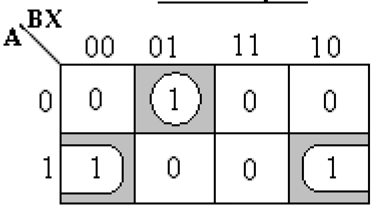
$S_A = A'B'X + A'BX'$
$= A'(B \oplus X)$

For R_A



$R_A = ABX' + AB'X'$

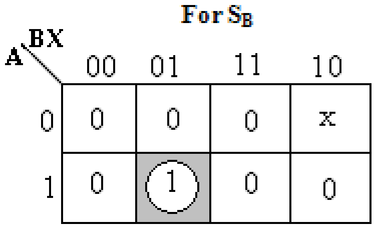
For Output



$Y = A'B'X + AX'$

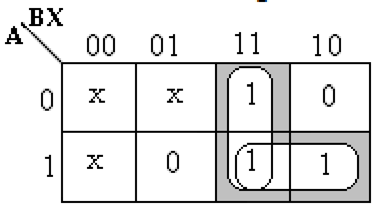
For Flip-flop B

For S_B



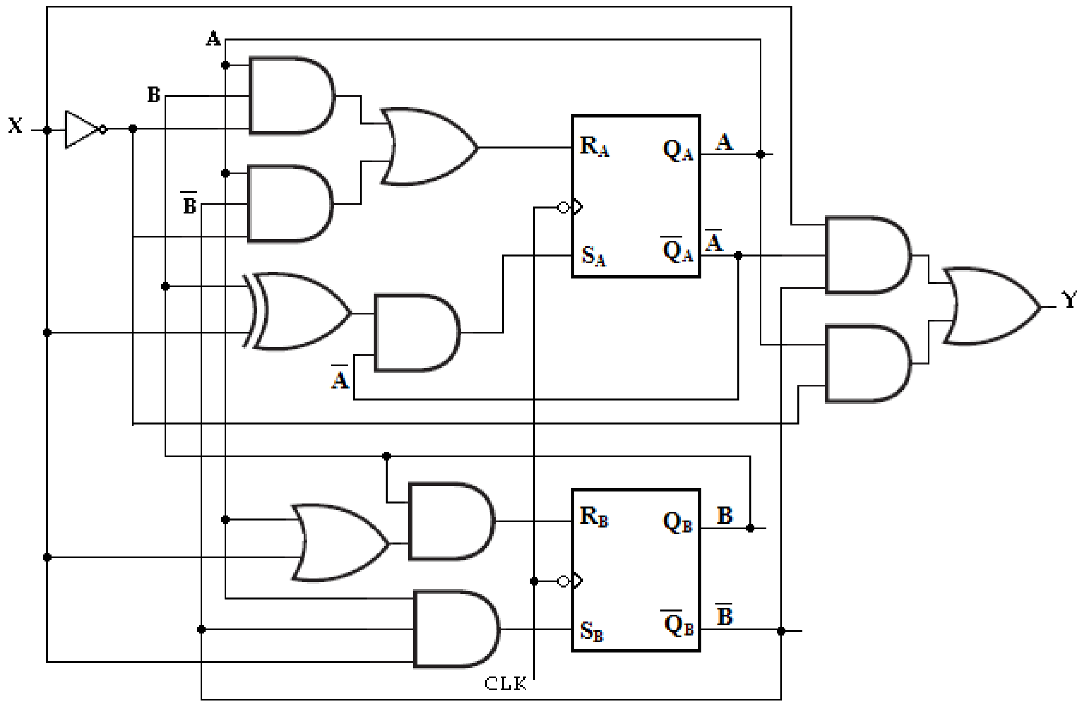
$S_B = AB'X$

For R_B



$R_B = AB + BX$

WiththeseFlip-Flopinputfunctionsandcircuitoutputfunctionwecandrawthelogicdiagramasfollows.



iii) DesignusingJKFlip-Flops:

UsingtheexcitationtableforJKFlip-Flop,wecandeterminetheexcitationtableforthegivencircuitas,

PresentState	NextState	Inputs	
Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

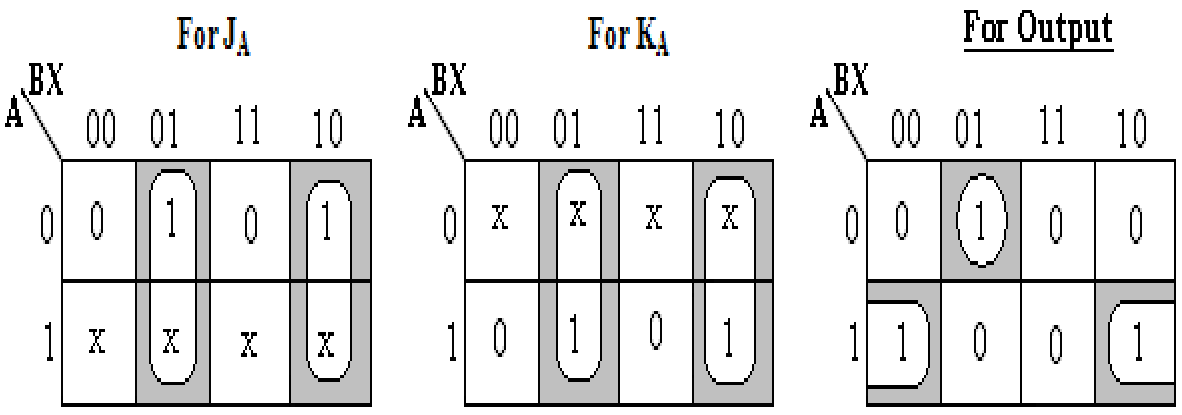
ExcitationtableforJKFlip-Flop

Present state		Input	Nextstate		Flip-FlopInputs				Output
A	B	X	A	B	J_A	K_A	J_B	K_B	Y
0	0	0	0	0	0	x	0	x	0
0	0	1	1	0	1	x	0	x	1
0	1	0	1	1	1	x	x	0	0
0	1	1	0	0	0	x	x	1	0
1	0	0	1	0	x	0	0	x	1
1	0	1	0	1	x	1	1	x	0
1	1	0	0	0	x	1	x	1	1
1	1	1	1	0	x	0	x	1	0

Circuitexcitationtable

K-mapSimplification:

For Flip-flop A

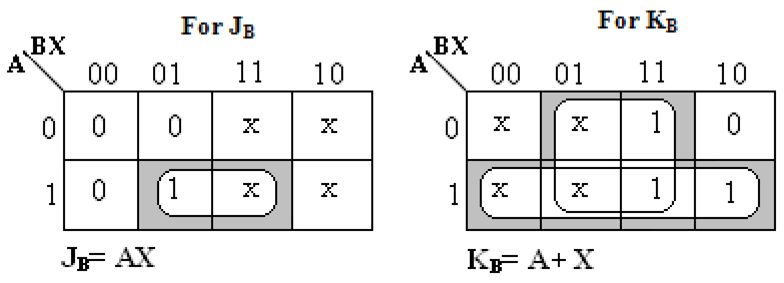


$J_A = BX' + B'X$
$= B \oplus X$

$K_A = BX' + B'X$
$= B \oplus X$

$Y = A'B'X + AX'$

For Flip-flop B



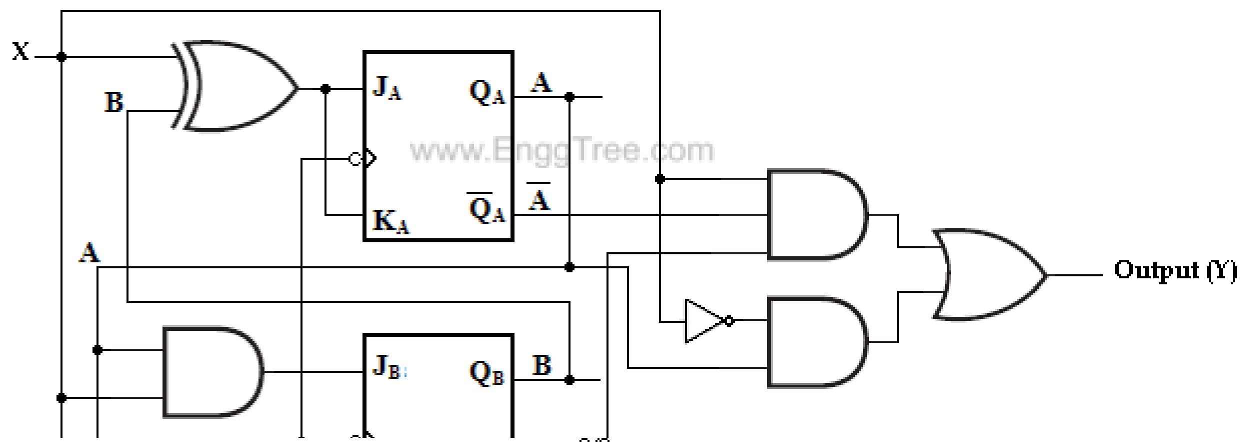
Theinputfunctionsfor,

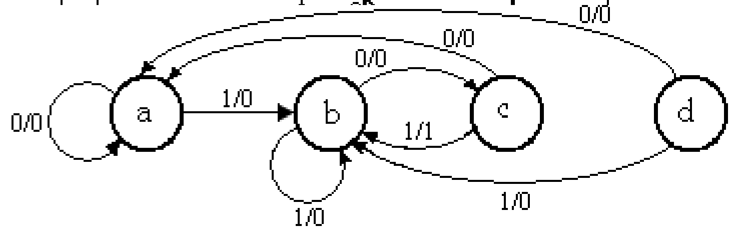
J_A=BX'+B'X **J_B=AX**
=B⊕X

K_A=BX'+B'X **K_B=A+X**
=B⊕X

Circuitoutputfunction,**Y=AX'+A'B'X**
WiththeseFlip-Flopinputfunctionsandcircuitoutputfunctionwecandrawthelogicdiagramasfollows.

Logicdiagramofgivensequentialcircuitusing JKFlip-Flop





2. DesignaclockedsequentialmachineusingJKFlip-Flopsforthestatediagramshowninthefigure. Usestate reduction if possible. Makeproperstateassignment.

Soln:

State Table:

Presentstate	**Nextstate**		**Output**	
	X=0	**X=1**	**X=0**	**X=1**
a	a	b	0	0
b	c	b	0	0
c	a	b	0	1
d	a	b	0	0

Fromtheabovestatetable**a**and**d**generateexactlysamenextstateandsameoutput for every possible set of inputs. The state **a** and**d** go to next states **a** and **b**and have outputs 0 and 0 for x=0 and x=1 respectively. Therefore state **d**can beremoved and replaced by**a**. The finalreducedstate table isshownbelow.

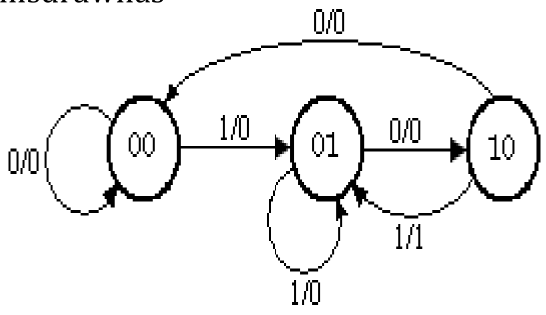
Presentstate	**Nextstate**		**Output**	
	X=0	**X=1**	**X=0**	**X=1**
a	a	b	0	0
b	c	b	0	0
c	a	b	0	1

ReducedStatetable

BinaryAssignment:

Now each state is assigned with binary values. Since there are three states,numberofFlip-Flopsrequiredistwoand2binarynumbersareassignedtothestates.a=00; b=0; andc=10

Thereducedstatediagramisdrawnas

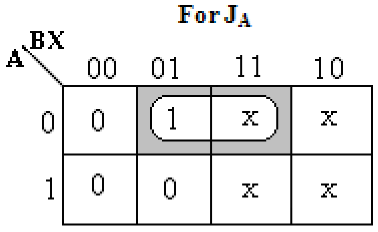


ReducedStateDiagram

K-mapSimplification:

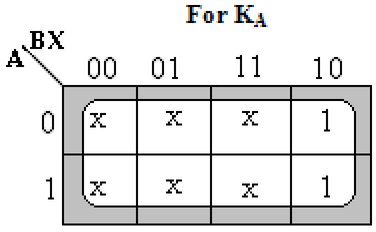
For Flip-flop A

For J_A



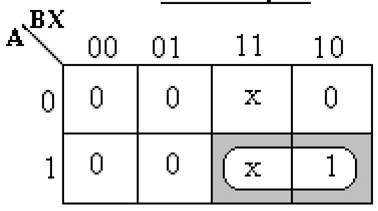
$J_A = X'B$

For K_A



$K_A = 1$

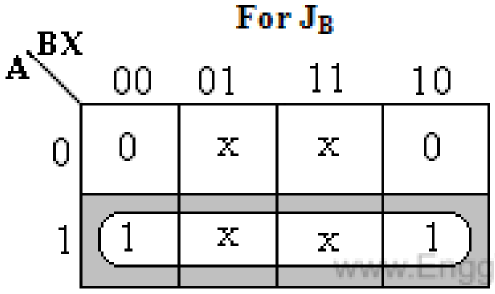
For Output



$Y = XA$

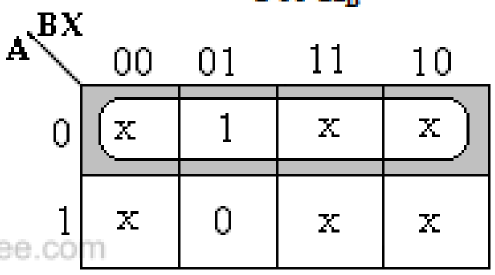
For Flip-flop B

For J_B



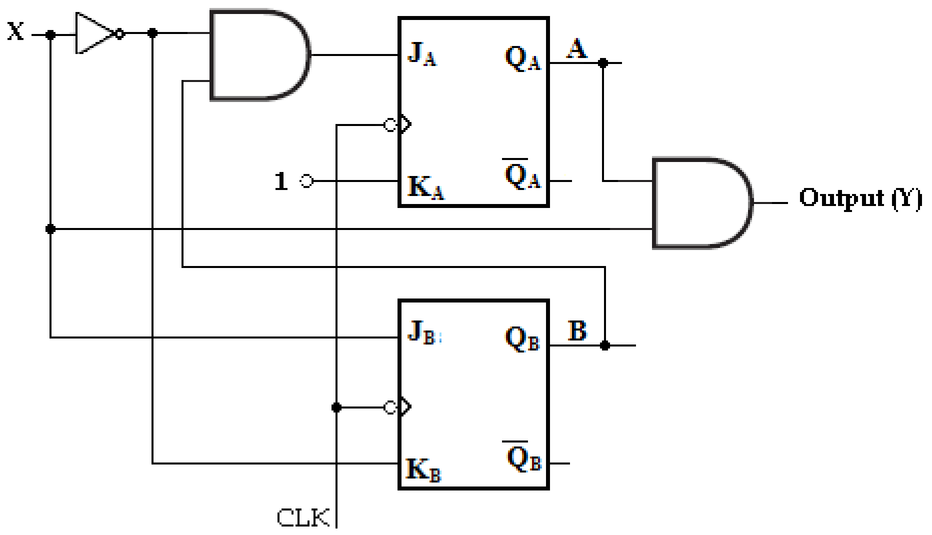
$J_B = X$

For K_B



$K_B = X'$

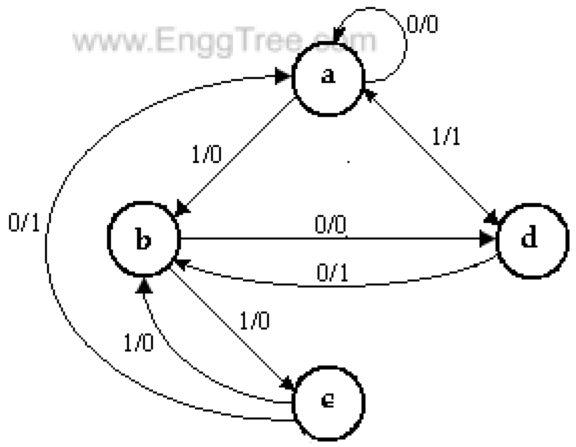
WiththeseFlip-Flopinputfunctionsandcircuitoutputfunctionwecandrawthelogicdiagramasfollows.



3. DesignaclockedsequentialmachineusingTFlip-Flopsforthefollowingstatediagram. Use statereduction ifpossible.Alsouse straightbinary stateassignment.

Soln:

State Table:



Statetableforthegivenstatediagramis,

Presentstate	Nextstate		Output	
	X=0	X=1	X=0	X=1
a	a	b	0	0
b	d	c	0	0
c	a	b	1	0
d	b	a	1	1

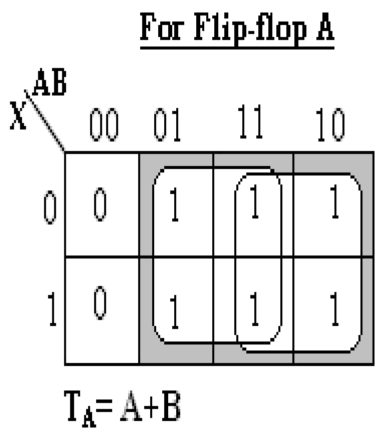
Eventhoughaandcarehavingsamenextstatesforinput X=0andX=1,astheoutputsar enotsamestatereductionisnotpossible.

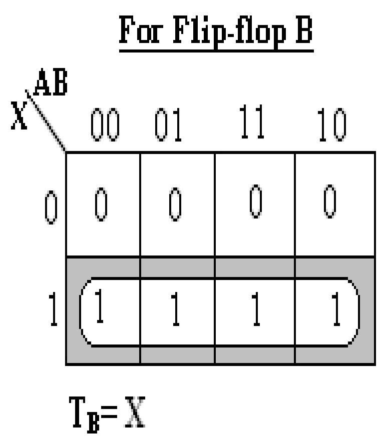
StateAssignment:

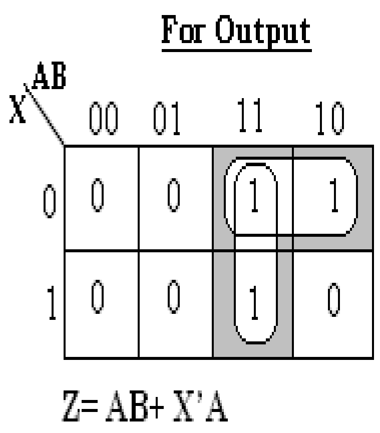
Usestraightbinaryassignmentsasa=00,b=01,c=10andd=11,thetransitiontabl eis,

Input	Presentstate		Nextstate		Flip-Flop Inputs		Output
X	A	B	A	B	T_A	T_B	Y
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	1	0	1
0	1	1	0	1	1	0	1
1	0	0	0	1	0	1	0
1	0	1	1	0	1	1	0
1	1	0	0	1	1	1	0
1	1	1	0	0	1	1	1

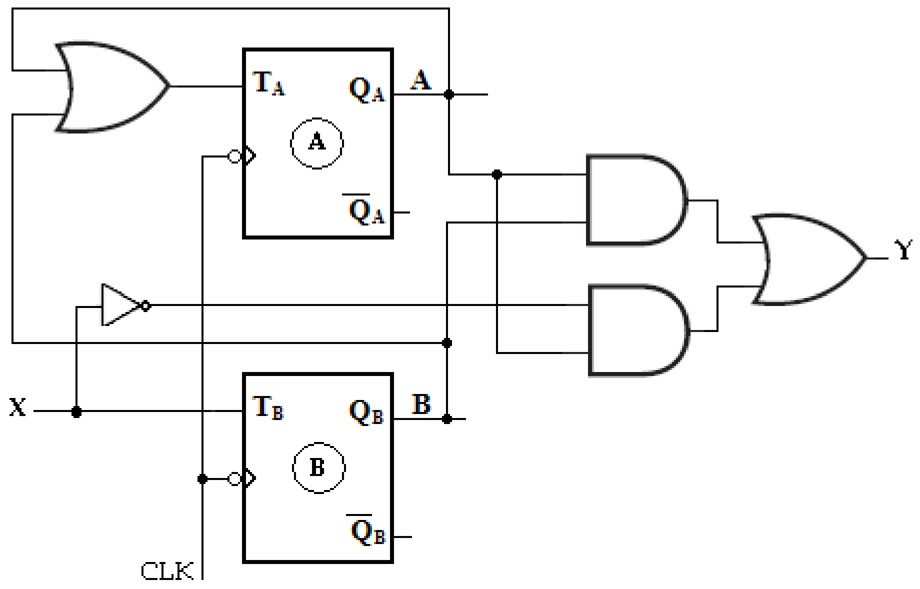
K-mapsimplification:







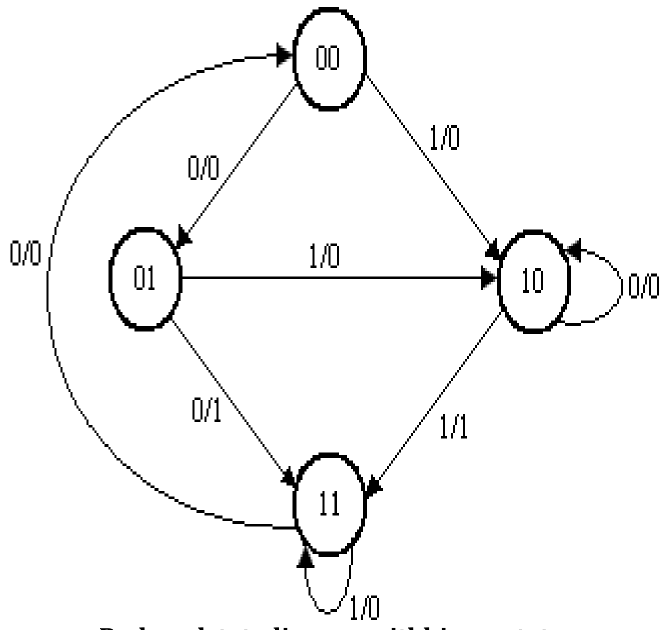
LogicDiagram:



STATEASSIGNMENT:

Insequentialcircuits,thebehaviorofthecircuitisdefinedintermsofitsinputs,presentstates,nextstatesandoutputs.Togeneratedesirednextstateatparticular present state and inputs, it is necessary to have specificFlip-Flop inputs.TheseFlip-FlopinputsaredescribedbyasetofBooleanfunctionscalledFlip-Flopinputfunctions.

To determine the Flip-Flop functions, it is necessary to represent states in thestate diagram using binaryvalues instead of alphabets. This procedure is known as***stateassignment***.



Reducedstatediagramwithbinarystates

Rulesforstateassignments

Therearetwobasicrulesformakingstateassignments.

Rule1:

States havingthe**same**NEXTSTATES foragiveninputconditionshouldhaveassignmentswhichcanbegroupedintologicallyadjacentcellsinaK-map.

Rule2:

StatesthataretheNEXTSTATESofasinglestateshouldhaveassignmentwhichcan begroupedintologicallyadjacentcellsinaK-map.

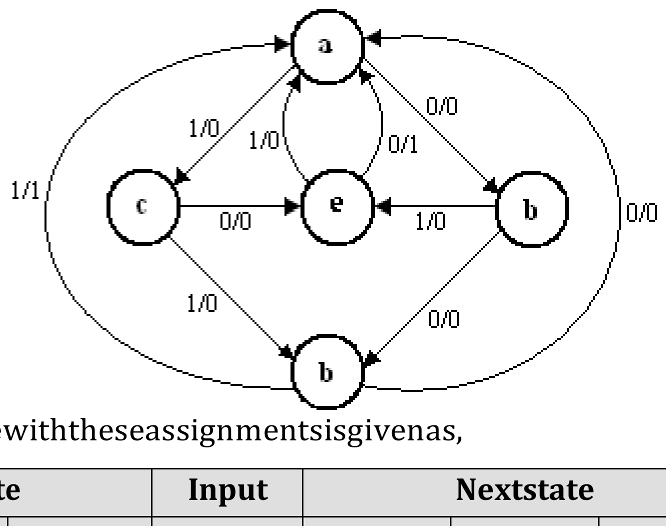
Presentstate	**Nextstate**		**Output**	
	X=0	**X=1**	**X=0**	**X=1**
00	01	10	0	0
01	11	10	1	0
10	10	11	0	1
11	00	11	0	0

Statetablewithassignmentstates

StateAssignmentProblem:

1. Design a sequential circuit for a state diagram shown below. Use stateassignmentrulesforassigningstatesandcomparetherequiredcombinationalcircuitwithrandomstateassignment.

Usingrandomstateassignmentweassign,a=0 00,b=001,c=010,d=011ande=100.

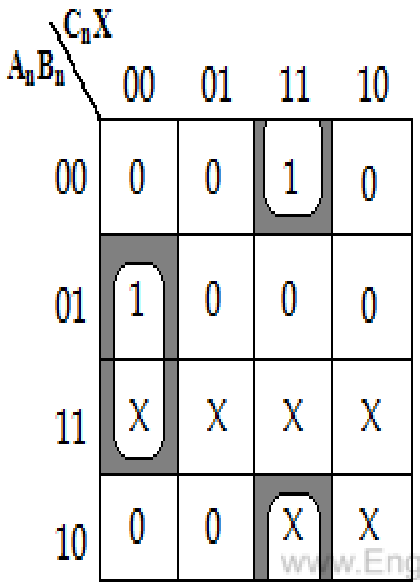


Theexcitationtablewiththeseassignmentsisgivenas,

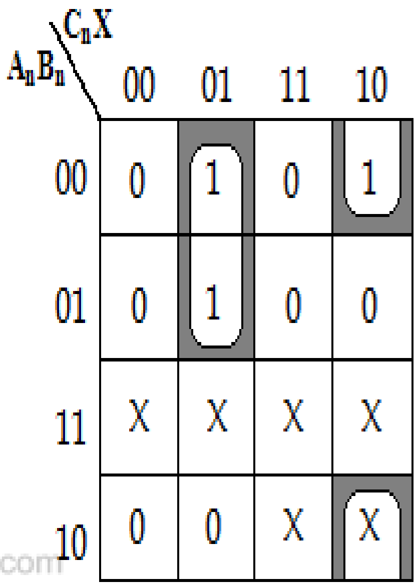
Presentstate			**Input**	**Nextstate**			**Output**
A_n	**B_n**	**C_n**	**X**	**A_{n+1}**	**B_{n+1}**	**C_{n+1}**	**Z**
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	0	0
0	1	0	1	0	1	1	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1

1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

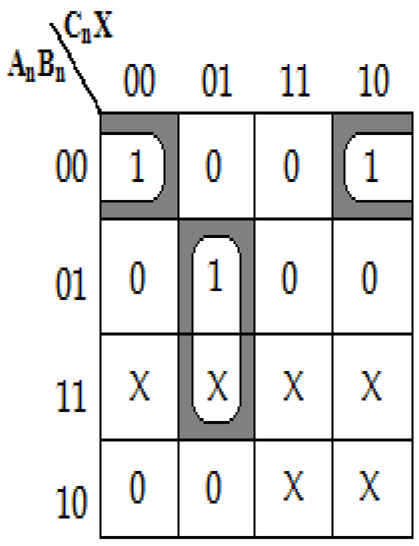
K-mapSimplification:



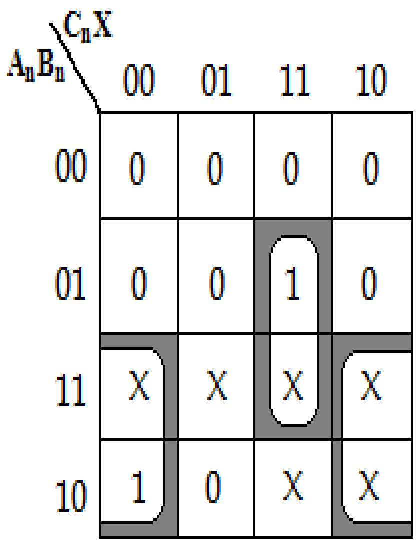
$D_A = B_n\bar{C}_n\bar{X} + \bar{B}_nC_nX$



$D_B = \bar{A}_n\bar{C}_nX + \bar{B}_nC_n\bar{X}$



$D_C = \bar{A}_n\bar{B}_n\bar{X} + B_n\bar{C}_nX$



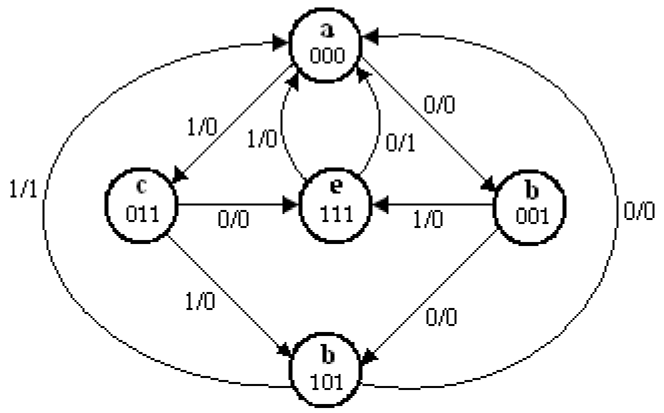
$Z = B_nC_nX + A_n\bar{X}$

Therandomassignmentsrequire:

7threeinputANDfunctions1t
woinputANDfunction
4twoinputORfunctions

12gateswith31inputs

Now,wewillapplythestateassignmentrulesandcomparetheresults.



StatediagramafterapplyingRules1and2

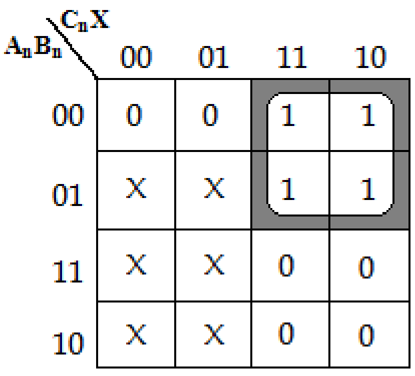
Rule1saysthat: eanddmustbeadjacent,and
bandcmustbeadjacent.

Rule2saysthat: eanddmustbeadjacent,and
bandcmustbeadjacent.

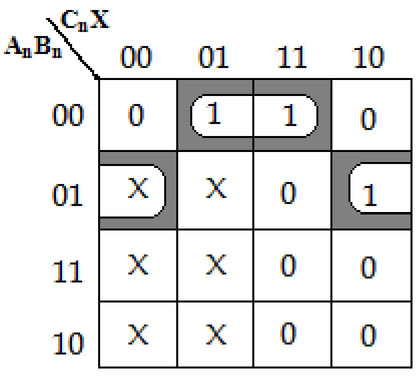
ApplyingRule1,Rule2tothestatediagramwegetthestateassignmentas,

Presentstate			Input	Nextstate			Output
A_n	B_n	C_n	X	A_{n+1}	B_{n+1}	C_{n+1}	Z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	1	0
0	0	1	0	1	0	1	0
0	0	1	1	1	1	1	0
0	1	0	0	x	x	x	x
0	1	0	1	x	x	x	x
0	1	1	0	1	1	1	0
0	1	1	1	1	0	1	0
1	0	0	0	x	x	x	x
1	0	0	1	x	x	x	x
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0

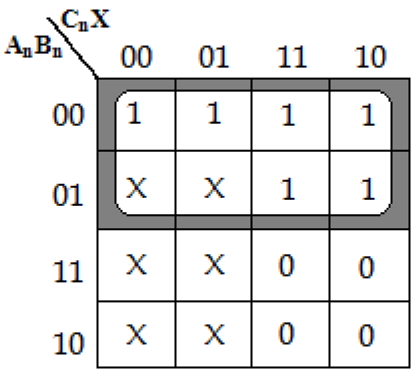
K-mapSimplification:



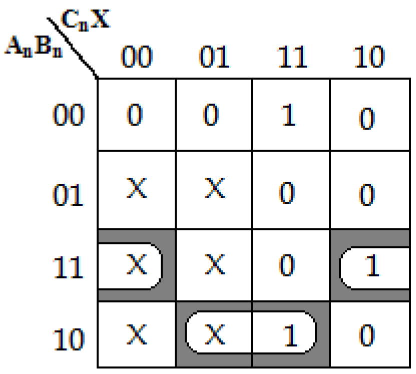
$A_{n+1} = D_A = \overline{A}_n C_n$



$B_{n+1} = D_B = \overline{A}_n\overline{B}_n X + \overline{A}_n B_n \overline{X}$



$C_{n+1} = D_C = \overline{A}_n$



$Z = A_n B_n \overline{X} + A_n \overline{B}_n X$

ThestateassignmentsusingRule1and2require:4thre
einputANDfunctions
1twoinputANDfunction2t
woinputORfunctions

7gateswith18inputs

ThusbysimplyapplyingRules1and2goodresultshavebeenachieved.

SYNCHRONOUSCOUNTERS

Flip-Flops can be connected together to perform counting operations. Such agroup of Flip- Flops is a **counter**. The number of Flip-Flops used and the way inwhich they are connected determine the number of states (called the modulus) andalsothespecificsequenceofstatesthatthecountergoesthroughduringeachcomplete cycle.

Countersareclassifiedintotwobroadcategoriesaccordingtothewaytheyareclocked:

- Asynchronous counters,
- Synchronouscounters.

In asynchronous (ripple) counters, the first Flip-Flop is clocked by the externalclockpulseandtheneachsuccessiveFlip-FlopisclockedbytheoutputoftheprecedingFlip-Flop.

In synchronous counters, the clock input is connected to all of the Flip-Flops sothat they are clocked simultaneously. Within each of these two categories, countersare classified primarily by the type of sequence, the number of states, or the numberofFlip-Flopsinthecounter.

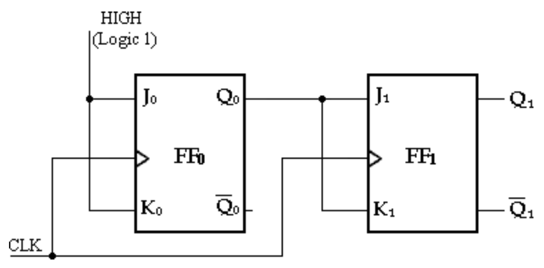
Theterm'synchronous'referstoeventsthathaveafixedtimerelationshipwith each other. In synchronous counter, the clock pulses are applied to all Flip-Flopssimultaneously.Hencethereisminimumpropagationdelay.

S.No	Asynchronous(ripple)counter	Synchronouscounter
1	AlltheFlip-Flopsare not clockedsimultaneously.	All the Flip-Flopsare clocked simultaneously.
2	The delay times of all Flip-Flopsareadded.Thereforet hereisconsiderable propagation delay.	Thereisminimumpropagationdelay.
3	Speedofoperationislow	Speedofoperation ishigh.
4	Logiccircuitisverysimple evenformorenumberofstates.	Design involvescomplexlogiccircuit asnumberofstateincreases.
5	Minimumnumbersoflogic devicesareneeded.	Thenumberoflogicdevicesismore thanripplecounters.
6	Cheaperthansynchronous counters.	Costlierthanripplecounters.

2-BitSynchronousBinaryCounter

Inthiscountertheclocksignalisconnectedinparalleltoclockinputsofboth the Flip-Flops (FF$_0$and FF$_1$). The output of FF$_0$is connected to J$_1$and K$_1$inputs of thesecondFlip-Flop(FF$_1$).

2-BitSynchronousBinaryCounter



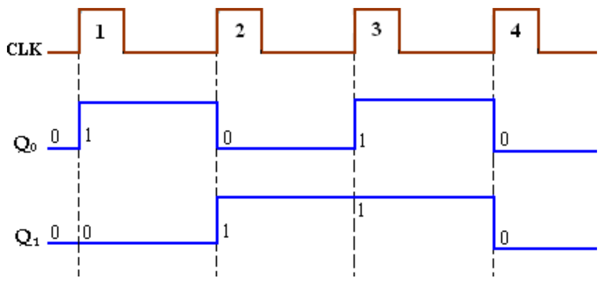
Assume that the counter is initially in the binary 0 state: i.e., both Flip-Flopsare RESET. When the positive edge of the first clock pulse is applied, FF$_0$ will togglebecause J$_0$= k$_0$= 1, whereas FF$_1$ output will remain 0 because J$_1$= k$_1$= 0. After the firstclockpulseQ$_0$=1andQ$_1$=0.

When the leading edge of CLK2 occurs, FF$_0$will toggle and Q$_0$will go

LOW.Since FF_1has a HIGH (Q_0= 1) on its $J_1$and K_1inputs at the triggering edge of thisclockpulse,theFlip-FloptogglesandQ_1goesHIGH.Thus,afterCLK2, Q_0=0andQ_1=1.

WhentheleadingedgeofCLK3occurs,FF_0againtogglestotheSETstate(Q_0 = 1), and FF_1 remains SET (Q_1 = 1) because its J_1 and K_1 inputs are both LOW (Q_0 = 0).Afterthistriggeringedge,Q_0=1andQ_1=1.

Finally, at the leading edge of CLK4, Q_0 and Q_1 go LOW because they bothhave a toggle condition on their J_1 and K_1 inputs. The counter has now recycled to itsoriginalstate,Q_0=Q_1=0.



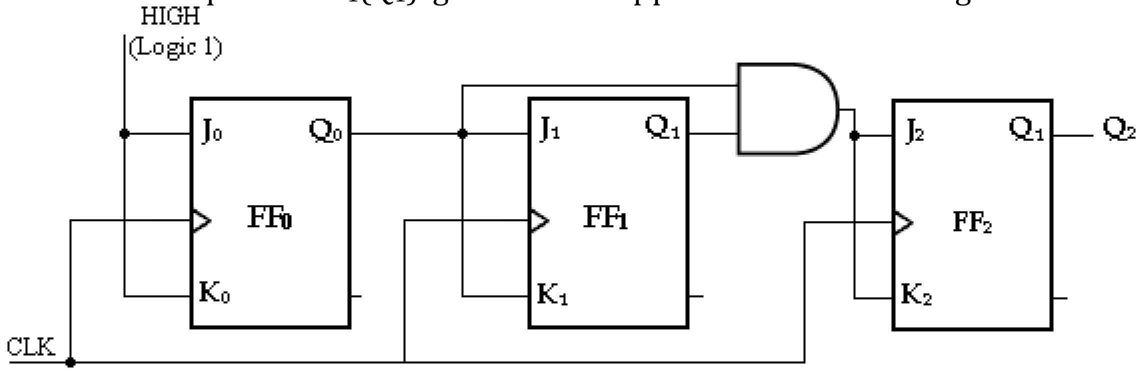
Timingdiagram

3-BitSynchronousBinaryCounter

A 3 bit synchronous binary counter is constructed with three JK Flip-Flopsand an AND gate. The output of FF_0 (Q_0) changes on each clock pulse as the counterprogresses from its original state to its final state and then back to its original state.To produce this operation, FF_0 must be held in the toggle mode by constant HIGH,onitsJ_0andK_0inputs.

3-BitSynchronousBinaryCounter

The output of FF_1(Q_1) goes to the opposite state following each time Q_0=



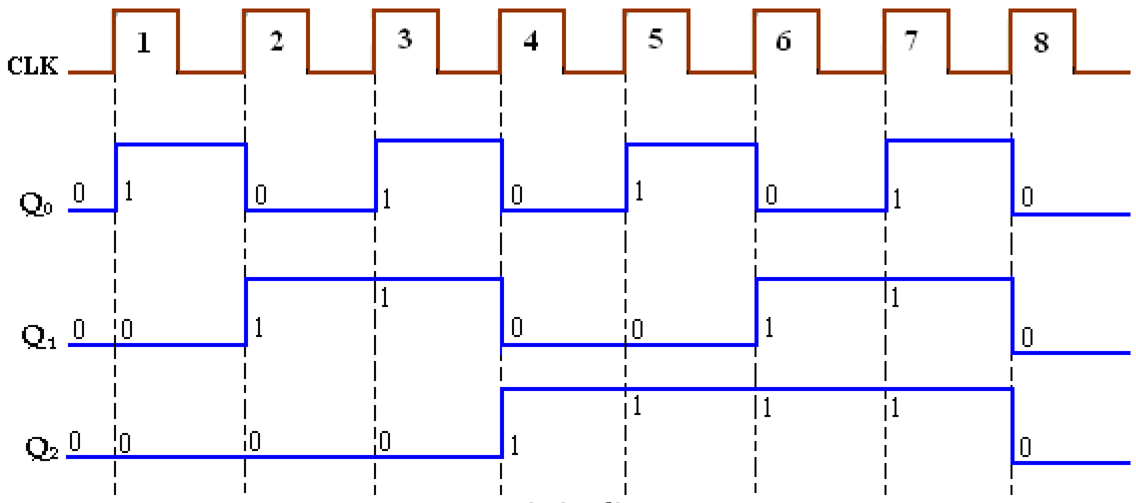
1.This change occurs at CLK2, CLK4, CLK6, and CLK8. The CLK8 pulse causes thecounter to recycle. To produce this operation, Q_0 is connected to the J_1 and K_1 inputsof FF_1. When Q_0= 1 and a clock pulse occurs, FF_1 is in the toggle mode and thereforechanges state. When Q_0= 0, FF_1 is in the no-change mode and remains in its presentstate.

The output of FF_2 (Q_2) changes state both times; it is preceded by the uniquecondition in which both Q_0 and Q_1 are HIGH. This condition is detected by the ANDgateandappliedtotheJ_2andK_2inputsofFF_3.WheneverbothoutputsQ_0=Q_1=1,

the output of the AND gate makes the J_2= K_2= 1 and FF_2toggles on the followingclock pulse. Otherwise, the $J_2$and K_2inputs of FF2 are held LOW by the AND gateoutput,FF_2doesnotchangestate.

CLOCKPulse	Q_2	Q_1	Q_0

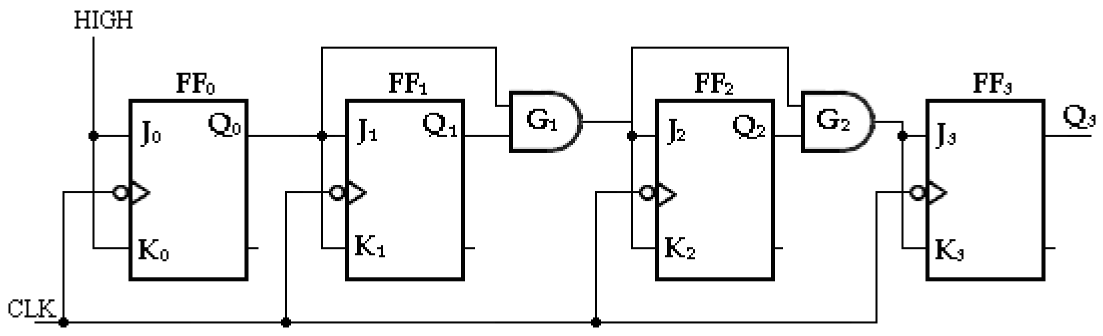
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8(recycles)	0	0	0



Timingdiagram

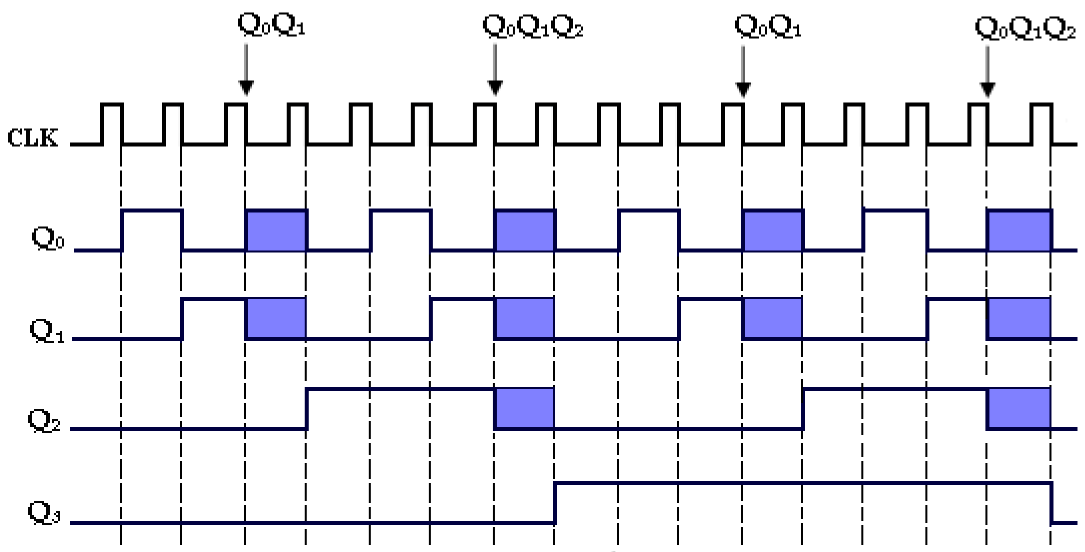
4-BitSynchronousBinaryCounter

This particular counter is implemented with negative edge-triggered Flip-Flops. The reasoning behind the J and K input control for the first three Flip- Flops isthe same as previously discussed for the 3-bit counter. For the fourth stage, the Flip-Flop has to change the state when Q_0= Q_1= Q_2= 1. This condition is decoded by ANDgateG_3.



4-BitSynchronousBinaryCounter

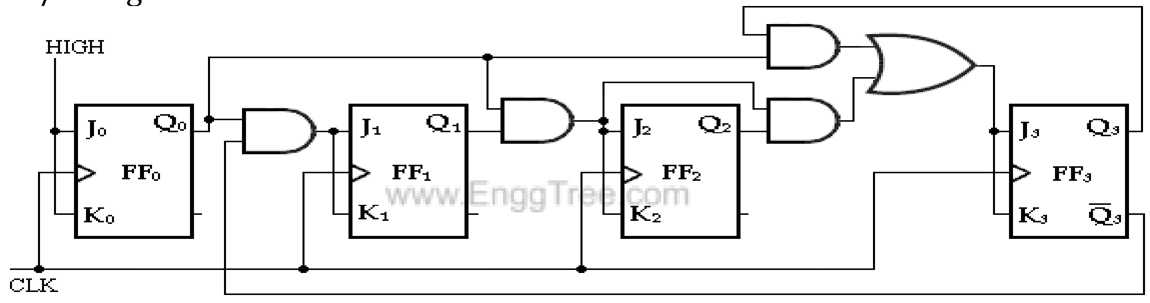
Therefore, when Q_0= Q_1= Q_2= 1, Flip-Flop FF_3 toggles and for all other times itisinano-changecondition.PointswheretheANDgateoutputsareHIGHareindicatedbytheshaded areas.



Timingdiagram

4-BitSynchronousDecadeCounter:(BCDCounter):

BCDdecadecounterhasasequencefrom0000to1001(9).After1001stateitmust recycle back to 0000 state. This counter requires four Flip-Flops and AND/ORlogicasshownbelow.



4-Bit Synchronous DecadeCounter

CLOCKPulse	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10(recycles)	0	0	0	0

- First,noticethatFF$_0$(Q_0)togglesoneachclockpulse,sothelogicequationforitsJ_0andK_0inputsis

$$J_0=K_0=1$$

ThisequationisimplementedbyconnectingJ_0andK_0toaconstantHIGHlevel.

- Next,noticefromtable,thatFF$_1$(Q_1)changesonthenextclockpulseeachtime Q0=1and Q3=0,sothelogicequationforthe$J_1$and K_1inputsis

$$J_1=K_1=Q_0Q_3'$$

ThisequationisimplementedbyANDingQ_0andQ_3andconnectingthegateoutputtotheJ_1andK_1inputsofFF_l.

- Flip-Flop2(Q_2)changesonthenextclockpulseeachtimebothQ_0=Q_1=1.This requiresaninputlogicequationasfollows:

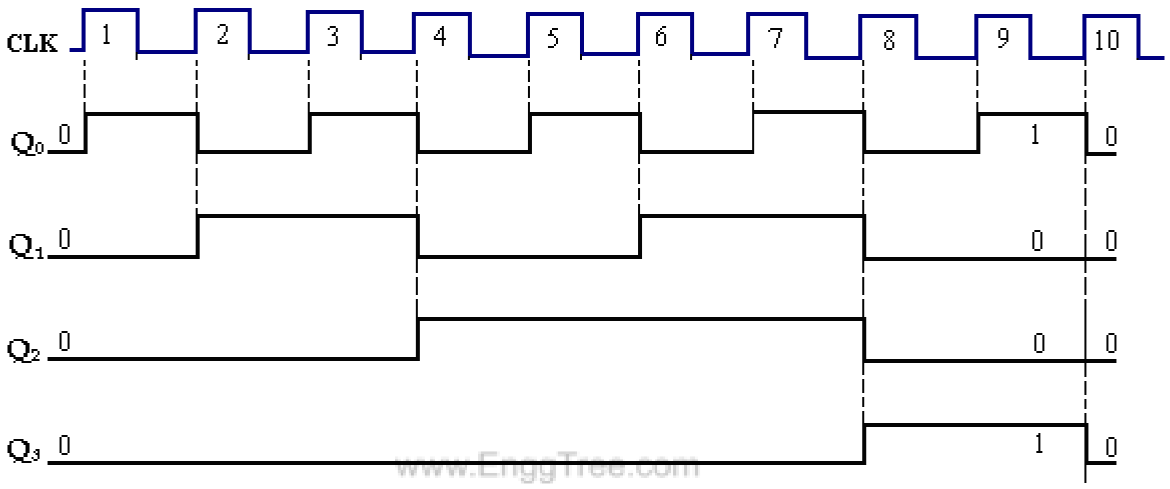
$$J_2=K_2=Q_0Q_1$$

ThisequationisimplementedbyANDingQ_0andQ_1andconnectingthegateoutputtotheJ_2andK_2inputsofFF_3

- Finally, FF_3(Q_3) changes to the opposite state on the next clock pulse eachtime Q_0 = 1, Q_1 = 1, and Q_2 = 1 (state 7), or when Q_0 = 1 and Q_1 = 1 (state 9).Theequationforthisisasfollows:

$$J_3=K_3=Q_0Q_1Q_2+Q_0Q_3$$

ThisfunctionisimplementedwiththeAND/ORlogicconnectedtotheJ_3andK_3inputsofFF_3.



Timingdiagram

SynchronousUP/DOWNCounter

Anup/downcounterisabidirectionalcounter,capableofprogressingineitherdirectionthroughacertainsequence.A3-bitbinarycounterthatadvancesupwardthroughitssequence(0,1,2,3,4,5,6,7)andthencanbe reversedsothatit goesthroughthesequenceintheoppositedirection(7,6,5,4,3,2,1,0)isanillustrationofup/downsequentialoperation.

The complete up/down sequence for a 3-bit binary counter is shown in tablebelow.Thearrowsindicatethestate-to-statemovementofthecounterforbothitsUPand its DOWN modes of operation. An examination of Q_0 for both the up and downsequences shows that FF_0 toggles on each clock pulse. Thus, the $J_0$and K_0 inputs of$FF_0$are,

$$J_0=K_0=1$$

CLOCK PULSE	UP	Q_2	Q_1	Q_0	DOWN
0		0	0	0	
1		0	0	1	
2		0	1	0	
3		0	1	1	
4		1	0	0	
5		1	0	1	
6		1	1	0	
7		1	1	1	

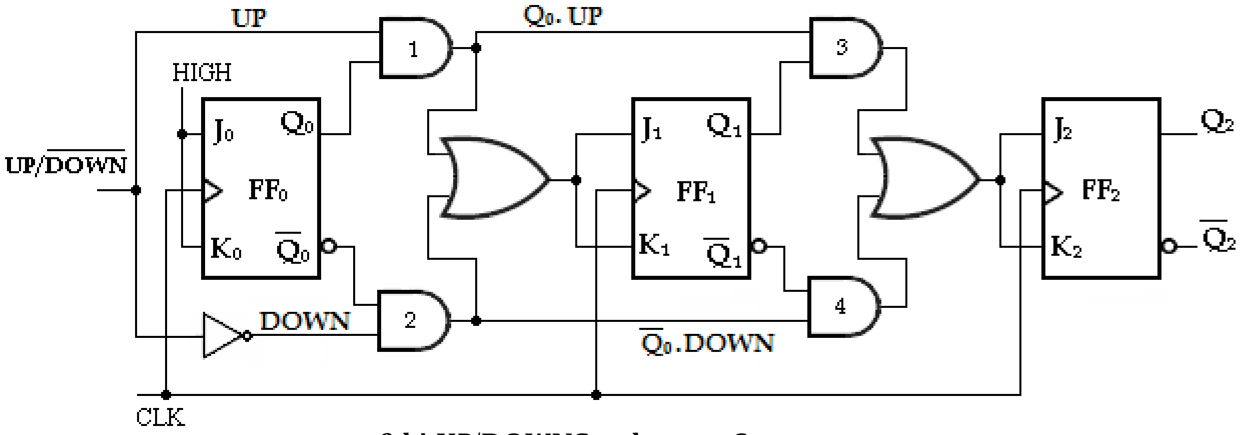
Toform a synchronous UP/DOWN counter, the control input (UP/DOWN)is used to allow either the normal output or the inverted output of one Flip-Flop totheJandKinputsofthenextFlip-Flop.WhenUP/DOWN=1,theMOD8counterwillcountfrom 000to111andUP/DOWN=0,it willcount from111 to 000.

WhenUP/DOWN=1,itwillenableANDgates1and 3 and disable ANDgates 2 and 4. This allows the Q0 and Q1 outputs through the AND gates to the J andKinputsofthefollowingFlip-Flops,sothecounter countsupaspulsesareapplied.

WhenUP/DOWN=0,thereverseactiontakesplace.

$$J_1=K_1=(Q_0.UP)+(Q_0'.DOWN)$$

$$J_2=K_2=(Q_0.Q_1.UP)+(Q_0'.Q_1'.DOWN)$$



3-bitUP/DOWNSynchronousCounter

MODULUS-N-COUNTERS

Thecounterwith‘n’Flip-FlopshasmaximumMODnumber2^n.Findthenumber of Flip-Flops (n) required for the desired MOD number (N) using theequation,

$$2^n \geq N$$

(i) Forexample,a3bitbinarycounterisaMOD8counter.Thebasiccountercanbe modified to produce MOD numbers less than 2^n by allowing the counter toskinthosearenormallypart ofcountingsequence.

$n=3$

$N=8$

$2^n=2^3=8=N$

(ii) MOD5Counter:

$2^n=N$

$2^n=5$

$2^2=4$lessthanN.

$2^3=8>N(5)$

Therefore,3Flip-Flopsarerequired.

(iii) MOD10Counter:

$2^n=N=10$

$2^3=8$lessthanN.

$2^4=16>N(10)$.

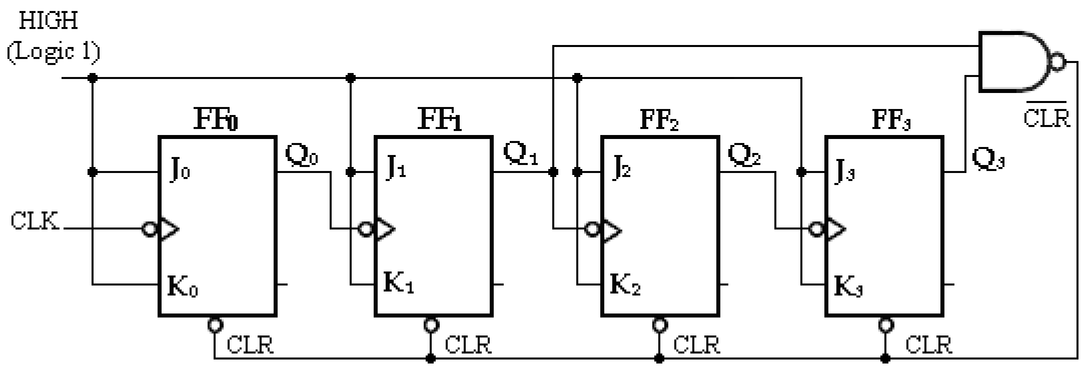
ToconstructanyMOD-Ncounter,thefollowingmethodscanbeused.

1. FindthenumberofFlip-Flops(n)requiredforthedesiredMODnumber(N)usingtheequation,
   $$2^n \geq N.$$
2. ConnectalltheFlip-Flopsasarequiredcounter.
3. FindthebinarynumberforN.
4. ConnectallFlip-FlopoutputsforwhichQ=1whenthecountisN,asinputs toNANDgate.
5. ConnecttheNANDgateoutputtotheCLRinputofeachFlip-Flop.

When the counter reaches Nthstate, the output of the NAND gate goes LOW,resettingallFlip-Flopsto0.ThereforethecountercountsfromOthroughN-1.

Forexample,MOD-10counterreachesstate10(1010).i.e.,$Q_3Q_2Q_1Q_0$=1010.TheoutputsQ_3andQ_1areconnectedtotheNANDgateandtheoutputoftheNANDgategoesLOWandresettingallFlip-Flopstozero.ThereforeMOD-10countercountsfrom 0000 to1001. And thenrecycles tothe zero value.

TheMOD-10countercircuitisshownbelow.



MOD-10(Decade)Counter

SHIFTREGISTERS:

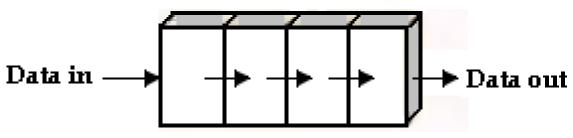
A register is simply a group of Flip-Flops that can be used to store a binarynumber.TheremustbeoneFlip-Flopforeachbitinthebinarynumber.Forinstance,aregister usedtostorean8-bitbinary number musthave8Flip-Flops.

TheFlip-Flopsmustbeconnectedsuchthatthebinarynumbercanbeentered(shifted) into the register and possibly shifted out. A group of Flip-Flops connectedtoprovide either orboth ofthese functionsis called a*shiftregister*.

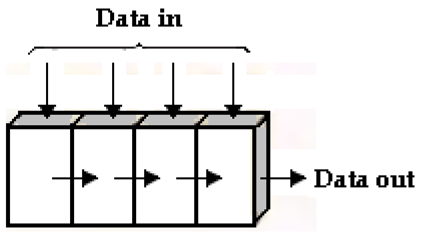
Thebitsinabinarynumber(data)canberemovedfromoneplacetoanother in either of two ways. The first method involves shifting the data one bit at a time ina serial fashion, beginning with either the most significant bit (MSB) or the leastsignificant bit (LSB). This technique isreferredto as *serialshifting*. The second methodinvolvesshiftingallthedatabitssimultaneouslyandisreferredtoas*parallel shifting*.

There are two ways to shift into a register (serial or parallel) and similarly twoways to shift the data out of the register. This leads to the construction of four basicregistertypes—

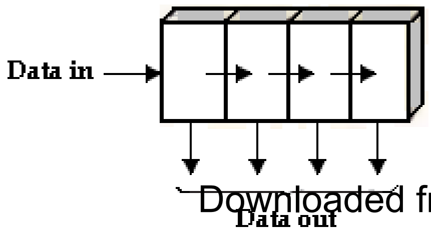
i. Serialin-serialout,
ii. Serialin-parallelout,
iii. Parallelin-serialout,
iv. Parallelin-parallelout.

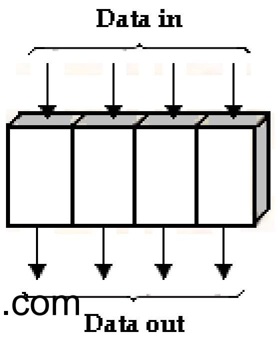


(i) **Serial in-serialout**



(iii)Parallelin-serialout



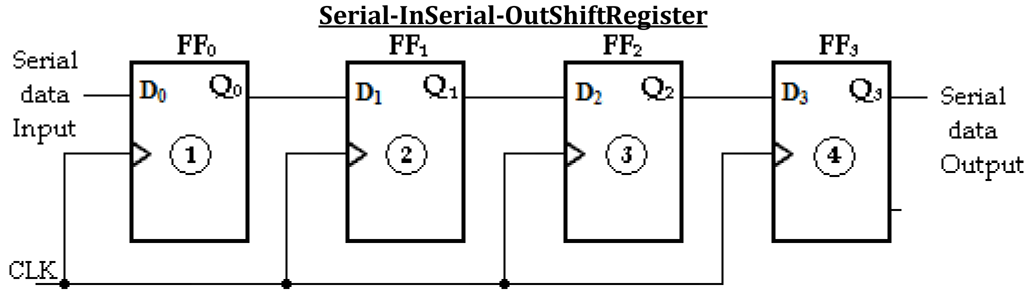


(iii)Serialin-parallelout **(iv)Parallelin-parallelout**

Serial-InSerial-OutShiftRegister:

Theserialin/serialoutshiftregisteracceptsdataserially,i.e.,onebitatatimeonasingleline.Itproducesthestoredinformationonitsoutputalsoinserialform.

Serial-InSerial-OutShiftRegister



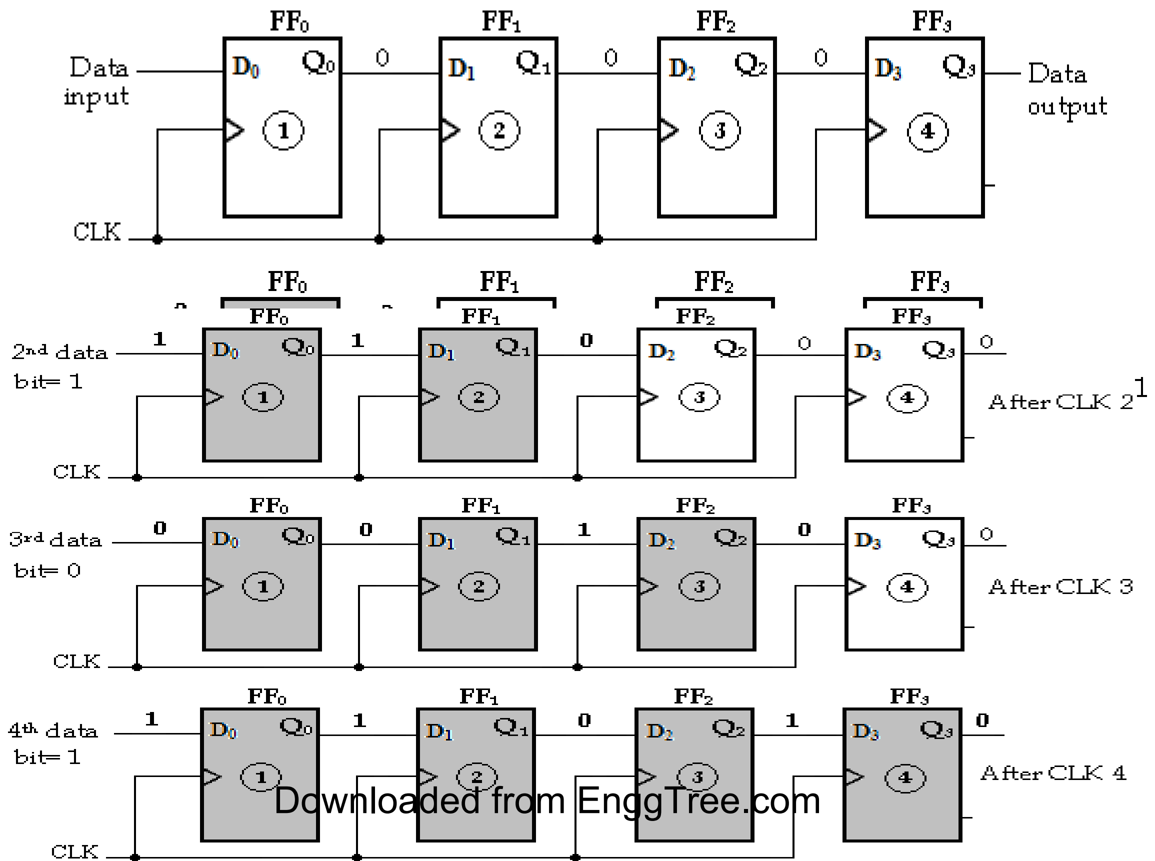
The entry of the four bits 1010 into the register is illustrated below, beginningwith the right-most bit. The register is initially clear. The 0 is put onto the datainputline,makingD=0forFF$_0$.Whenthefirstclockpulseisapplied,FF$_0$isreset,thusstoring the0.

Nextthesecondbit,whichisa1,isappliedtothedatainput,makingD=1forFF$_0$andD=0forFF$_1$becausetheDinputofFF$_1$isconnectedtotheQ$_0$output.When

thesecondclockpulseoccurs,the1onthedatainputisshiftedintoFF0,causingFF0toset;andthe0thatwasinFF0isshiftedintoFFl.

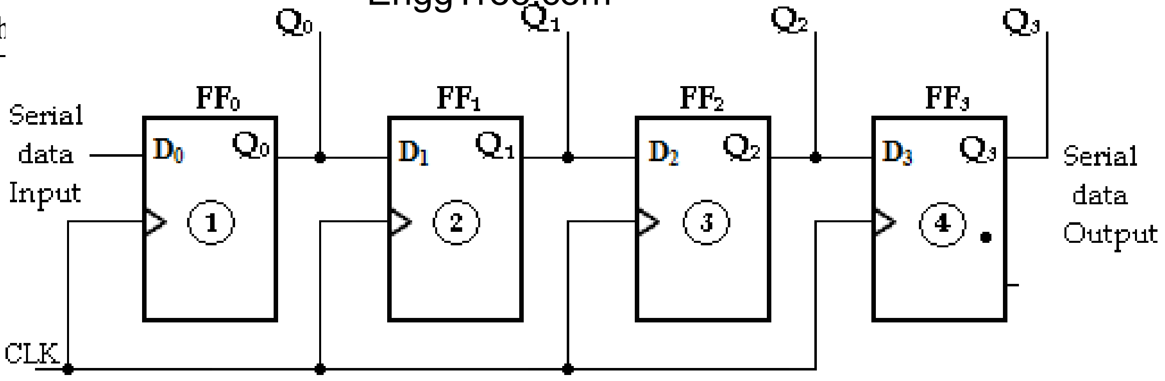
The third bit, a 0, is now put onto the data-input line, and a clock pulse isapplied. The 0 is entered into FF$_0$, the 1 stored in FF$_0$is shifted into FF$_l$, and the 0storedinFF$_1$isshiftedintoFF$_2$.

The last bit, a 1,is nowapplied to the data input, and a clock pulseis applied.This timethe 1 isentered intoFF0, the 0 stored in FF0 isshifted intoFFl, the 1 storedin FF1 isshiftedintoFF2, and the 0stored in FF2 is shiftedintoFF3. Thiscompletestheserialentryofthefourbitsintotheshiftregister,wheretheycanbestoredfor anylengthoftimeaslongastheFlip-Flopshavedcpower.



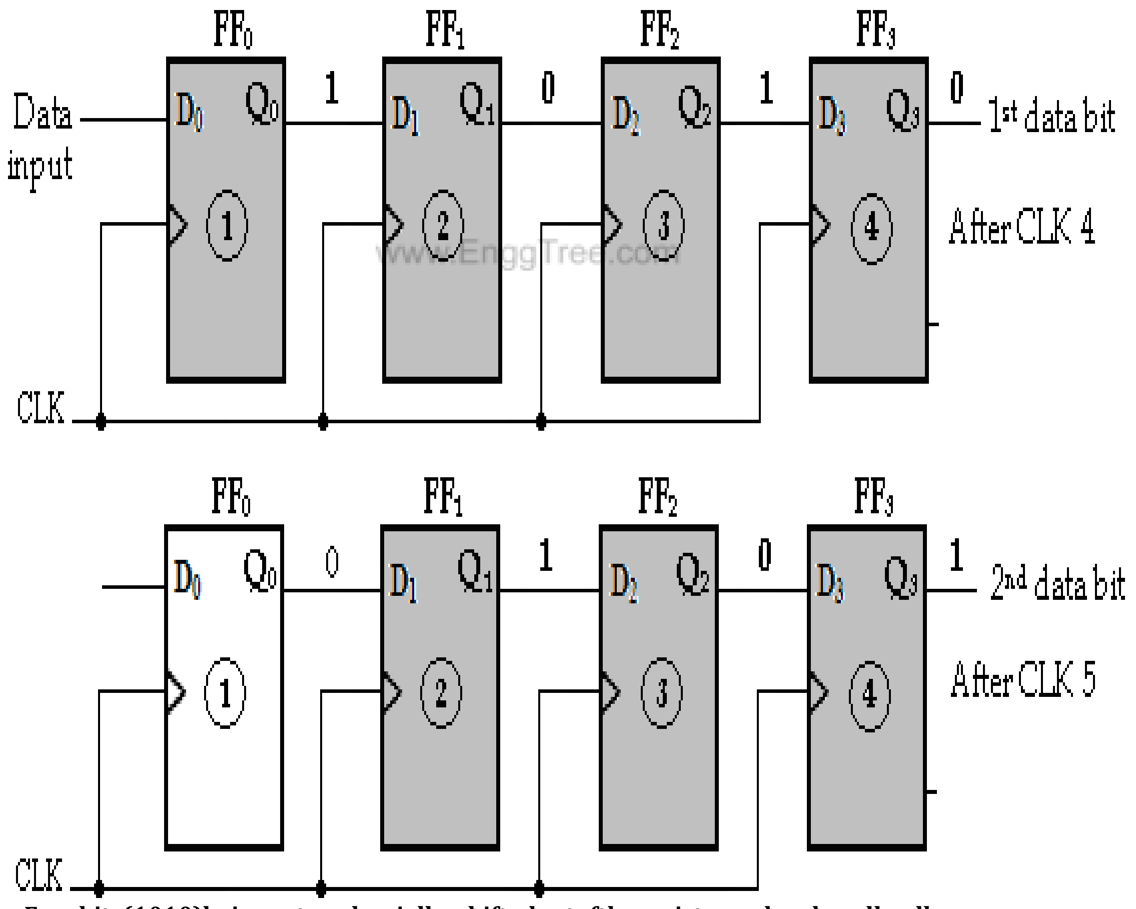
Fourbits(1010)beingenteredseriallyintotheregister

Synch



To get the data out of the register, the bits must be shifted out serially andtaken offtheQ3 output. AfterCLK4, theright-most bit, 0,appearson theQ3 output.

When clock pulse CLK5 is applied, the second bit appears on the Q3 output.Clock pulse CLK6 shifts the third bit to the output, and CLK7 shifts the fourth bit totheoutput.Whiletheoriginalfourbitsarebeingshiftedout,morebitscanbeshiftedin.All zerosareshownbeingshiftedout,morebitscanbeshiftedin.

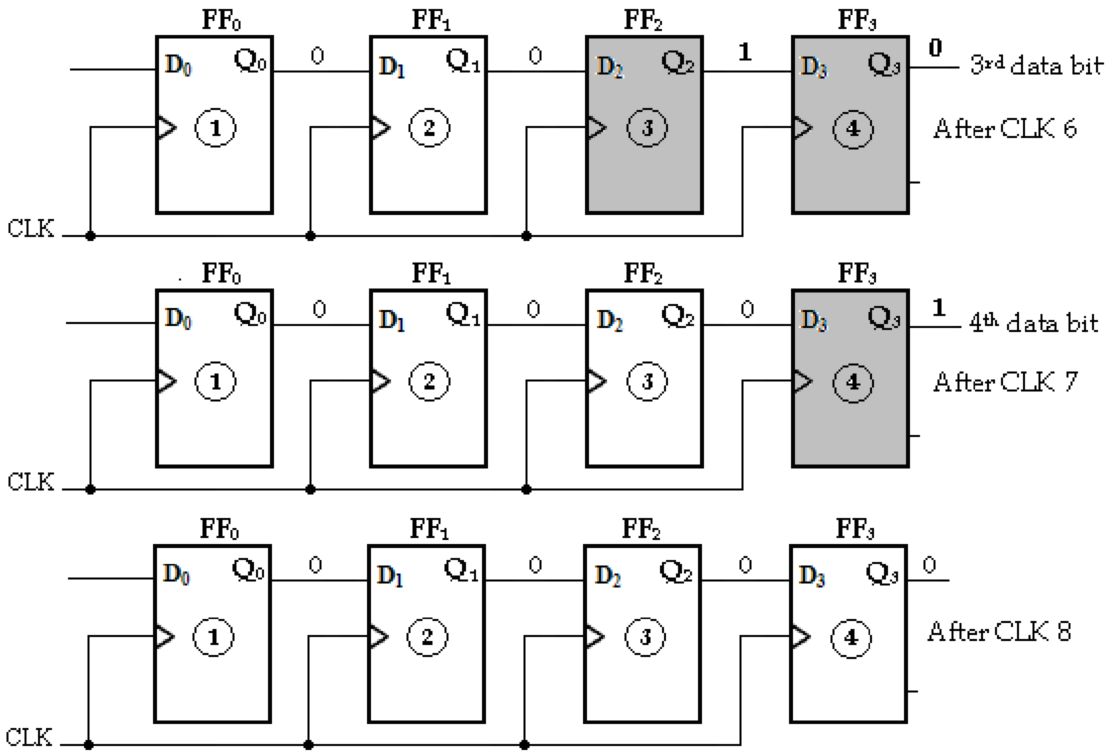


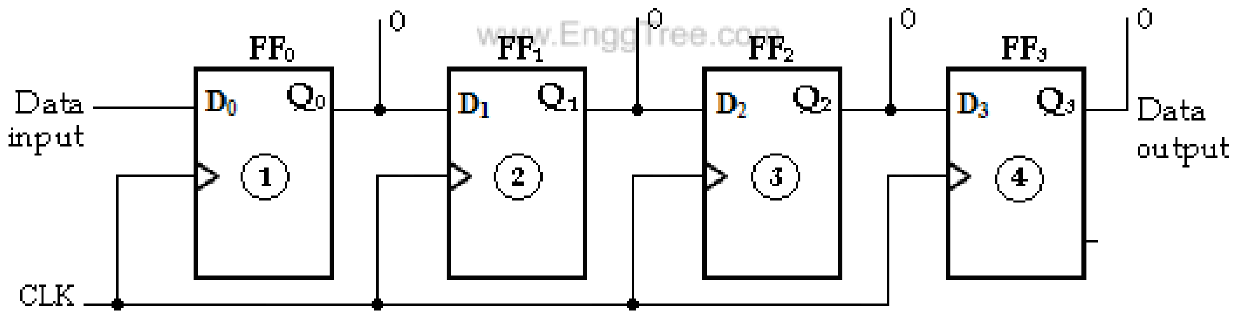
Fourbits(1010)beingenteredserially-shiftedoutoftheregisterandreplacedbyallzeros

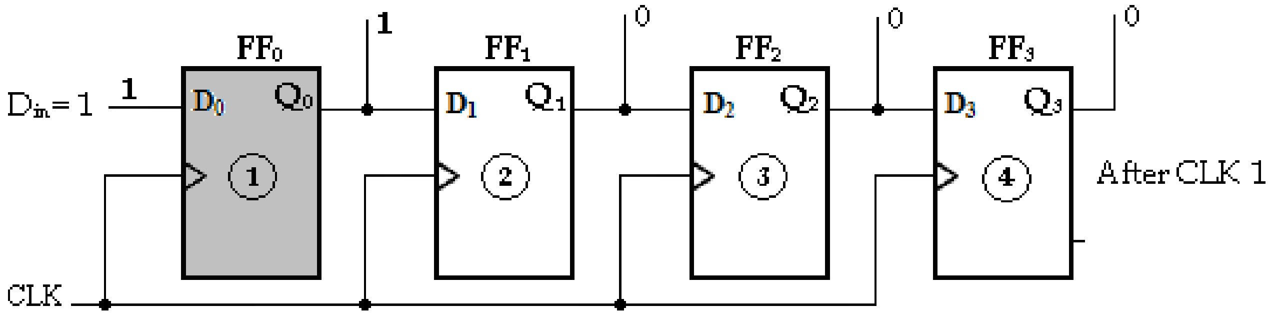
Serial-InParallel-OutShiftRegister:

In this shift register, data bits are entered into the register in the same asserial-inserial-outshiftregister.Buttheoutputistakeninparallel.Oncethedataarestored,eachbitappearsonitsrespectiveoutputlineandallbitsareavailablesimultaneouslyinsteadofonabit-by-bit.

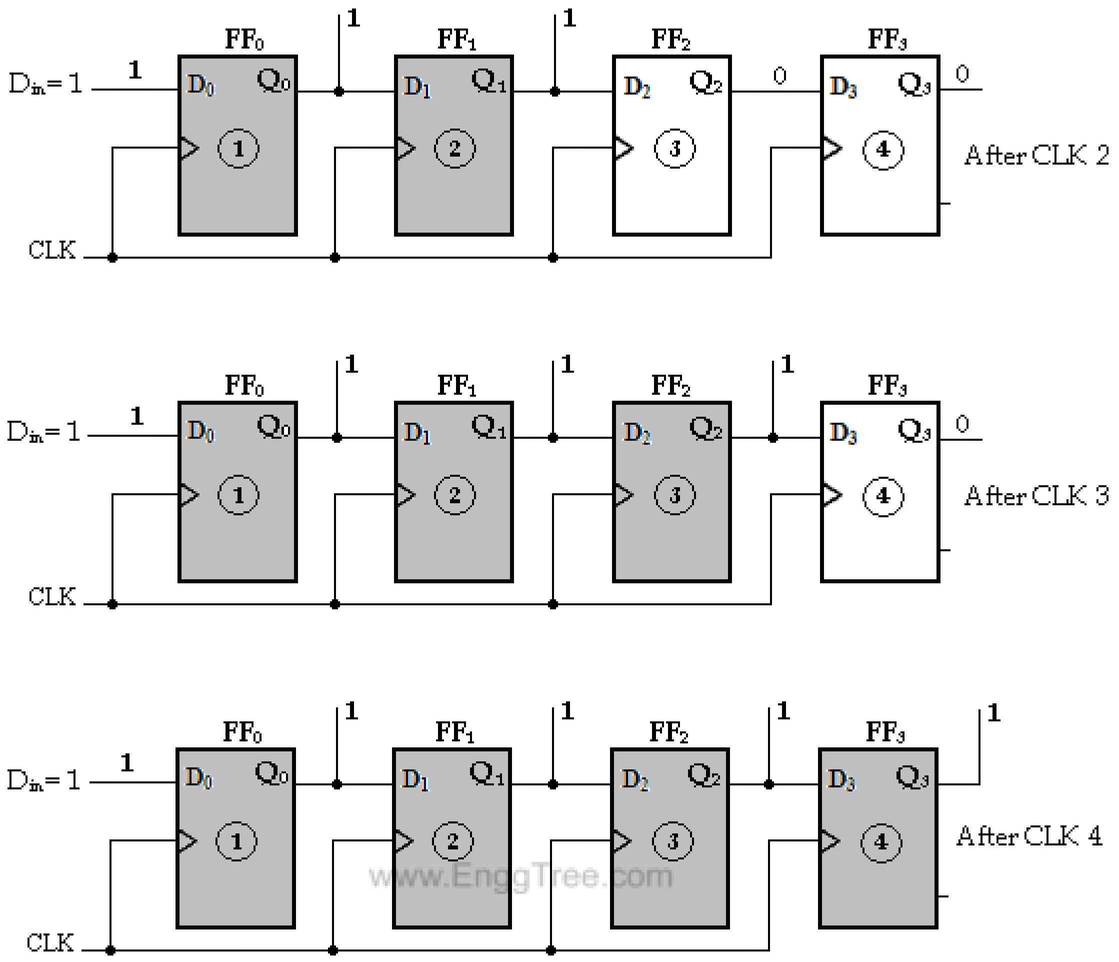
Serial-Inparallel-OutShiftRegister







X



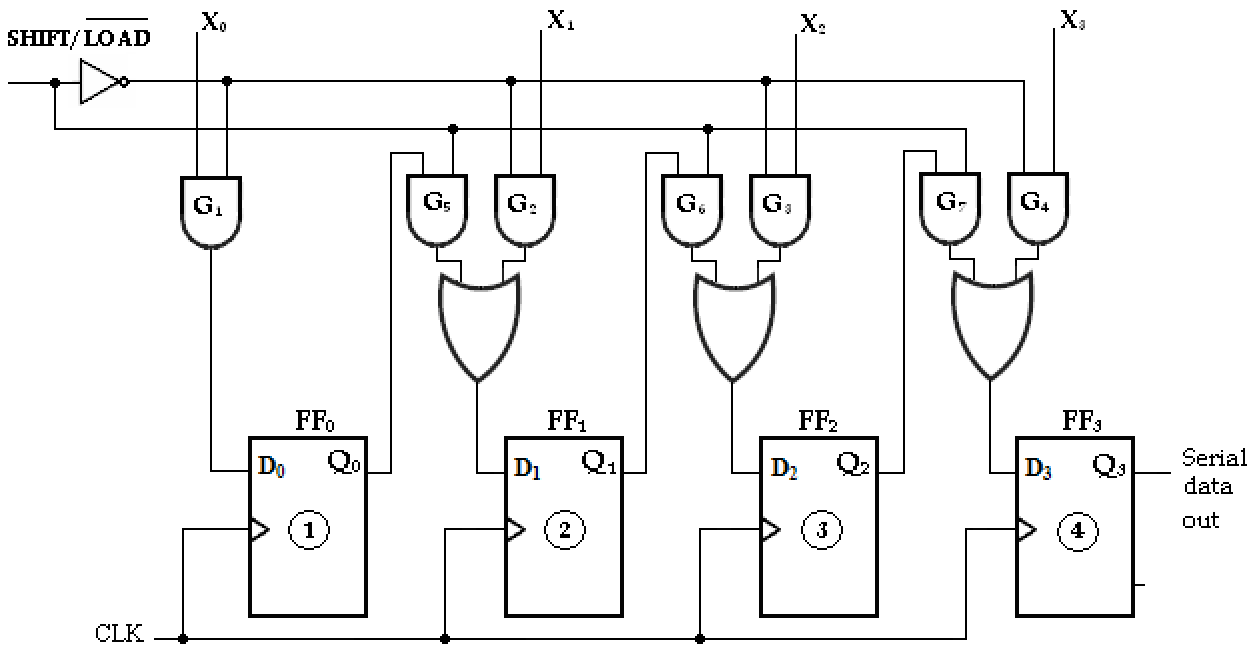
Fourbits(1111)beingseriallyenteredintotheregister

Parallel-InSerial-OutShiftRegister:

In this type, the bits are entered in parallel i.e., simultaneously into theirrespectivestagesonparallellines.

A 4-bit parallel-in serial-out shift register is illustrated below. There are fourdata input lines, X_0, X_1, $X_2$and $X_3$for entering data in parallel into the register.SHIFT/ LOAD input is the control input, which allows four bits of data to **load** inparallelintotheregister.

When SHIFT/LOADis LOW,gates G_1,G_2, $G_3$and G_4 are enabled, allowingeach data bit to be applied to the D input of its respective Flip-Flop. When a clockpulse is applied, the Flip-Flops with D = 1 will **set** and those with D = 0 will **reset,**therebystoringallfourbitssimultaneously.

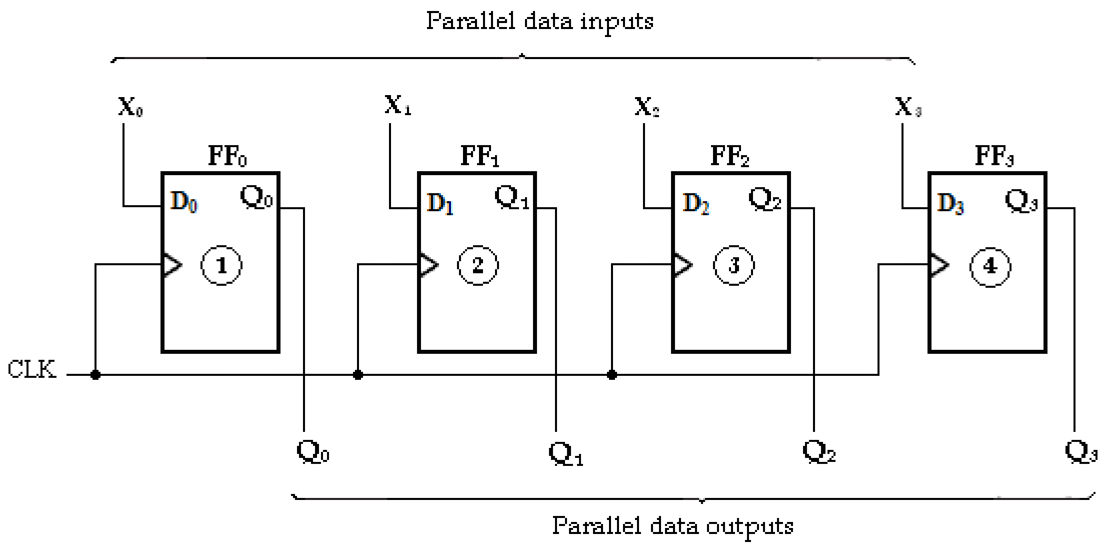


Parallel-InSerial-OutShiftRegister

WhenSHIFT/LOADisHIGH,gatesG_1,G_2,G_3andG_4aredisabledandgatesG_5, G_6 and G_7 are enabled, allowing the data bits to shift right from one stage to thenext. The OR gates allow either the normal shifting operation or the parallel data-entry operation, depending on which AND gates are enabled by the level on theSHIFT/LOADinput.

Parallel-InParallel-OutShiftRegister:

In this type, there is simultaneous entry of all data bits and the bits appear onparalleloutputssimultaneously.



Parallel-InParallel-OutShiftRegister

UNIVERSALSHIFTREGISTERS

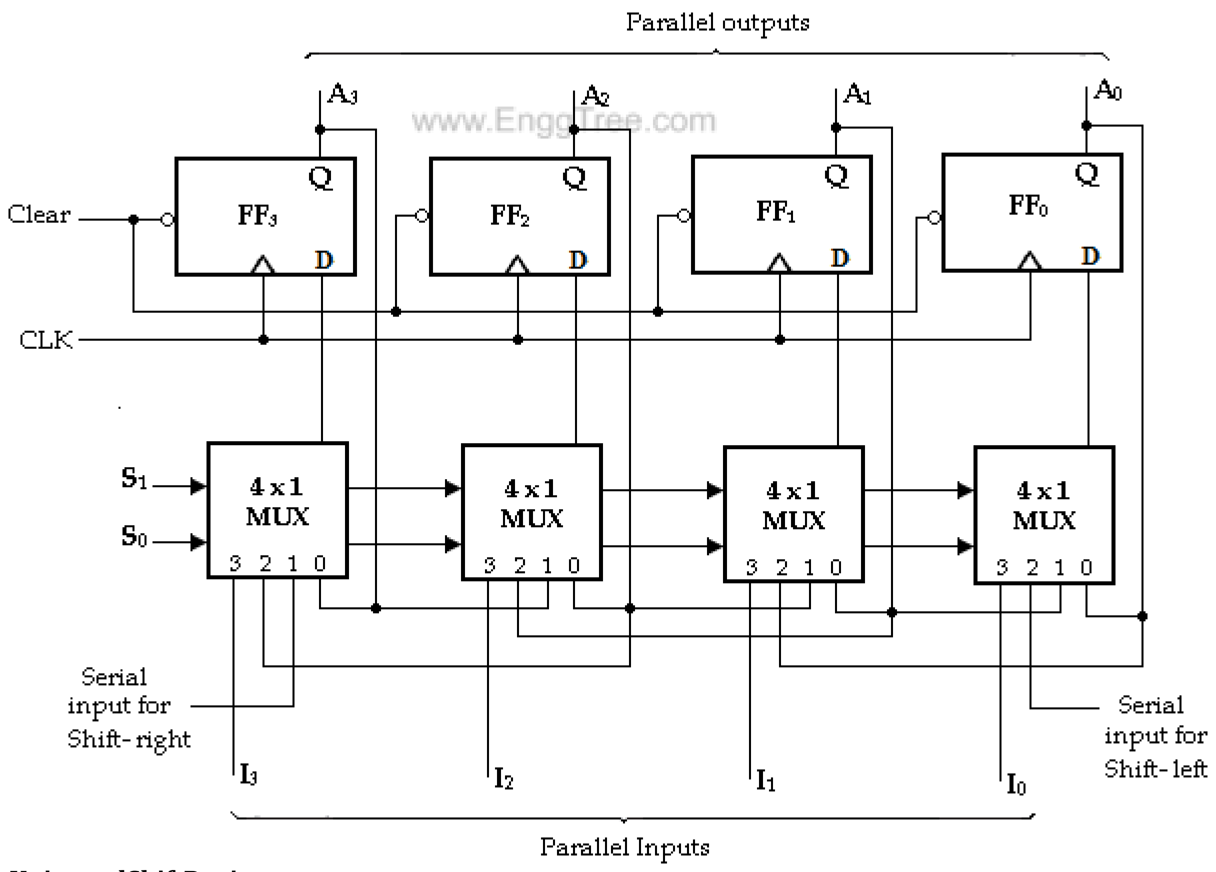
If the register has shift and parallel load capabilities, then it is called a shiftregister with parallel load or ***universal shift register***. Shift register can be used forconverting serial data to parallel data, and vice-versa. If a parallel load capability isaddedtoashiftregister,thedataenteredinparallelcanbetakenoutinserialfashionbyshiftingthedatastoredintheregister.

Thefunctionsofuniversalshiftregisterare:

- Aclearcontroltocleartheregisterto0.
- Aclockinputtosynchronizetheoperations.
- Ashift-rightcontroltoenabletheshiftrightoperationandtheserialinputandoutputlines associatedwiththeshiftright.
- Ashift-leftcontroltoenabletheshiftleftoperationandtheserialinputandoutputlinesassociatedwiththeshiftleft.
- Aparallel-loadcontroltoenableaparalleltransferandtheninputlinesassociatedwiththeparalleltransfer.
- ‘n’paralleloutputlines.
- Acontrollinethatleavestheinformationintheregisterunchangedeventhoughthe clockpulsesrecontinuouslyapplied.

It consists of four D-Flip-Flops and four 4 input multiplexers (MUX). S_0 and S_1arethetwoselectioninputsconnectedtoallthefourmultiplexers.Thesetwoselectioninputsareusedtoselectoneofthefourinputsofeachmultiplexer.

The input 0 in each MUX is selected when S_1S_0= 00 and input 1 is selectedwhen S_1S_0= 01. Similarly inputs 2 and 3 are selected when S_1S_0= 10 and S_1S_0= 11respectively.TheinputsS1andS0controlthemodeoftheoperationoftheregister.



4-BitUniversalShiftRegister

When S_1S_0= 00, the present value of the register is applied to the D-inputs of theFlip-Flops. This is done by connecting the output of each Flip-Flop to the 0 input ofthe respective multiplexer. The next clock pulse transfers into each Flip-Flop, thebinaryvalueisheldpreviously,andhencenochangeofstateoccurs.

When S_1S_0= 01, terminal 1 of the multiplexer inputs has a path to the D inputs oftheFlip-Flops.Thiscausesashift-rightoperationwiththelefterserialinputtransferredintoFlip-FlopFF_3.

When S_1S_0= 10, a shift-left operation results with the right serial input going intoFlip-FlopFF$_1$.

FinallywhenS$_1$S$_0$=11,thebinaryinformationontheparallelinputlines(I$_1$,I$_2$,I$_3$andI$_4$)aretransferredintotheregistersimultaneouslyduringthenextclockpulse.

Thefunctiontableofbi-directionalshiftregisterwithparallelinputsandparalleloutputsisshownbelow.

ModeControl		**Operation**
S$_1$	**S$_0$**	
0	0	No changeShi
0	1	ft-rightShift-
1	0	left
1	1	Parallelload

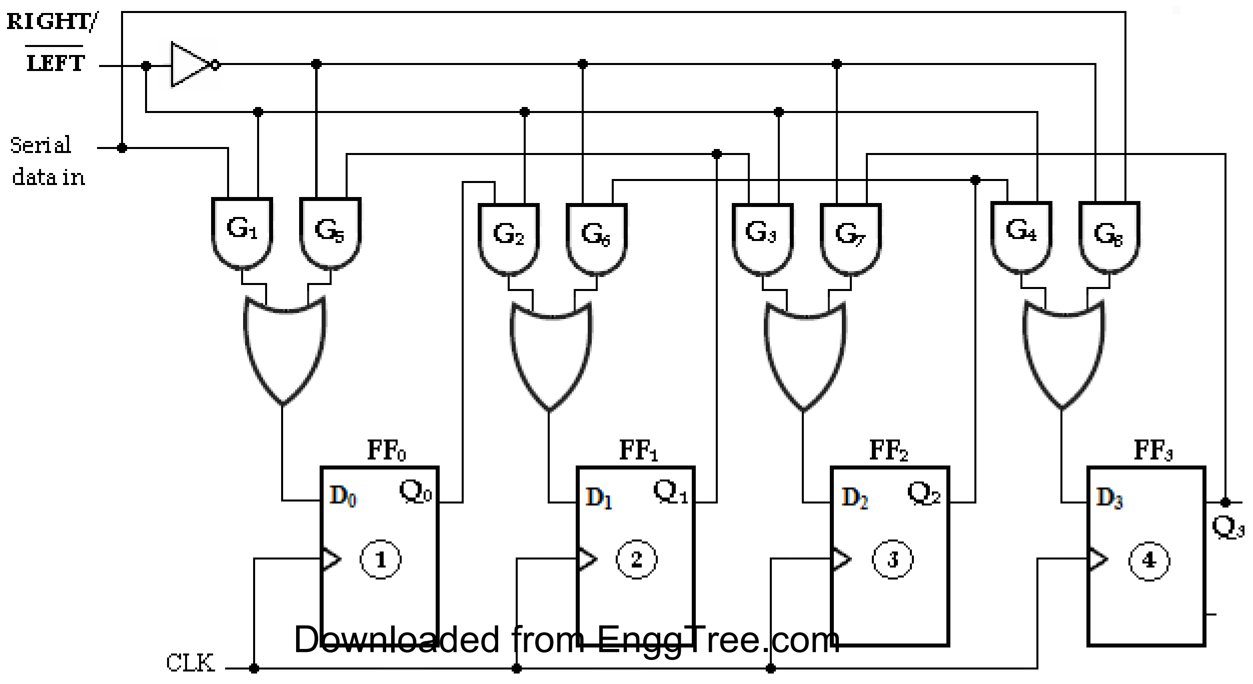
BI-DIRECTIONSHIFTREGISTERS:

A bidirectional shift register is one in which the data can be shifted either leftor right. It can be implemented by using gating logic that enables the transfer of adata bit from one stage to the next stage to the right or to the left depending on thelevelofacontrolline.

A4-bitbidirectionalshiftregisterisshownbelow.AHIGHontheRIGHT/LEFT control input allows data bits inside the register to be shifted to theright,andaLOW enablesdata bitsinsidetheregisterto beshifted tothe left.

When the RIGHT/LEFT control input is **HIGH**, gates G$_1$, G$_2$, G$_3$and G$_4$areenabled, and the state of the Q output of each Flip-Flop is passed through to the Dinput of the following Flip-Flop. When a clock pulse occurs, the data bits are shiftedoneplacetotheright.

When the RIGHT/LEFT control input is **LOW**, gates G$_5$, G$_6$, G$_7$and G$_8$areenabled, and the Q output of each Flip-Flop is passed through to the D input of thepreceding Flip-Flop. When a clock pulse occurs, the data bits are then shifted oneplacetotheleft.



4-bitbi-directionalshiftregister

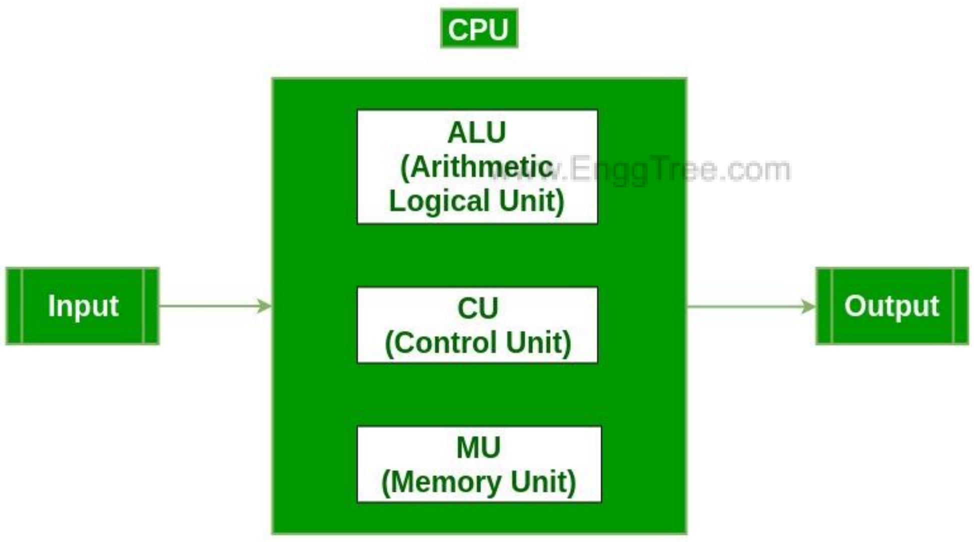
UNIT III

COMPUTER FUNDAMENTALS

FUNCTIONAL UNITS OF A DIGITAL COMPUTER:

Computer: A computer is a combination of **hardware and software** resources which integrate together and provides various functionalities to the user. Hardware are the physical components of a computer like the processor, memory devices, monitor, keyboard etc. while software is the set of programs or instructions that are required by the hardware resources to function properly.
There are a few basic components that aids the working-cycle of a computer i.e. the Input- Process- Output Cycle and these are called as the functional components of a computer. It needs certain input, processes that input and produces the desired output. The input unit takes the input, the central processing unit does the processing of data and the output unit produces the output. The memory unit holds the data and instructions during the processing.
Digital Computer: A digital computer can be defined as a programmable machine which reads the binary data passed as instructions, processes this binary data, and displays a calculated digital output. Therefore, Digital computers are those that work on the digital data.

Details of Functional Components of a Digital Computer



- **Input Unit :**The input unit consists of input devices that are attached to the computer. These devices take input and convert it into binary language that the computer understands. Some of the common input devices are keyboard, mouse, joystick, scanner etc.
- **Central Processing Unit (CPU) :** Once the information is entered into the computer by the input device, the processor processes it. The CPU is called the brain of the computer because it is the control center of the computer. It first fetches instructions from memory and then interprets them so as to know what is to be done. If required, data is fetched from memory or input device. Thereafter CPU executes or performs the required computation and then either stores the output or displays on the output device. The CPU has three main components which are responsible for different functions – Arithmetic Logic Unit (ALU), Control Unit (CU) and Memory registers
- **Arithmetic and Logic Unit (ALU) :** The ALU, as its name suggests performs mathematical calculations and takes logical decisions. Arithmetic calculations include addition, subtraction, multiplication and division. Logical decisions involve comparison of two data items to see which one is larger or smaller or equal.
- **Control Unit :** The Control unit coordinates and controls the data flow in and out of CPU and also controls all the operations of ALU, memory registers and also input/output units. It is also responsible for carrying out all the instructions stored in the program. It decodes the fetched instruction, interprets it and sends control signals to input/output devices until the required operation is done properly by ALU and memory.
- **Memory Registers :** A register is a temporary unit of memory in the CPU. These are used to store the data which is directly used by the processor. Registers can be of different sizes(16 bit, 32 bit, 64 bit and so on) and each register inside the CPU has

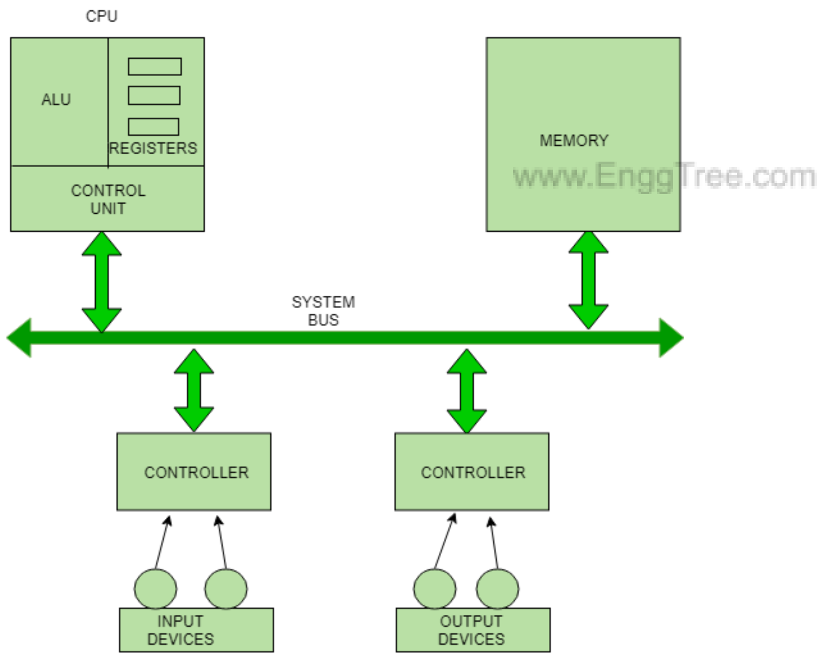
a specific function like storing data, storing an instruction, storing address of a location in memory etc. The user registers can be used by an assembly language programmer for storing operands, intermediate results etc. Accumulator (ACC) is the main register in the ALU and contains one of the operands of an operation to be performed in the ALU.

- **Memory :** Memory attached to the CPU is used for storage of data and instructions and is called internal memory The internal memory is divided into many storage locations, each of which can store data or instructions. Each memory location is of the same size and has an address. With the help of the address, the computer can read any memory location easily without having to search the entire memory. when a program is executed, it's data is copied to the internal memory and is stored in the memory till the end of the execution. The internal memory is also called the Primary memory or Main memory. This memory is also called as RAM, i.e. Random Access Memory. The time of access of data is independent of its location in memory, therefore this memory is also called Random Access memory (RAM). Read this for different types of RAMs
- **Output Unit :** The output unit consists of output devices that are attached with the computer. It converts the binary data coming from CPU to human understandable form. The common output devices are monitor, printer, plotter etc.

Interconnection between Functional Components

A computer consists of input unit that takes input, a CPU that processes the input and an output unit that produces output. All these devices communicate with each other through a common bus. A bus is a transmission path, made of a set of conducting wires over which data or information in the form of electric signals, is passed from one component to another in a computer. The bus can be of three types – Address bus, Data bus and Control Bus.

Following figure shows the connection of various functional components:



The address bus carries the address location of the data or instruction. The data bus carries data from one component to another and the control bus carries the control signals. The system bus is the common communication path that carries signals to/from CPU, main memory and input/output devices. The input/output devices communicate with the system bus through the controller circuit which helps in managing various input/output devices attached to the computer.

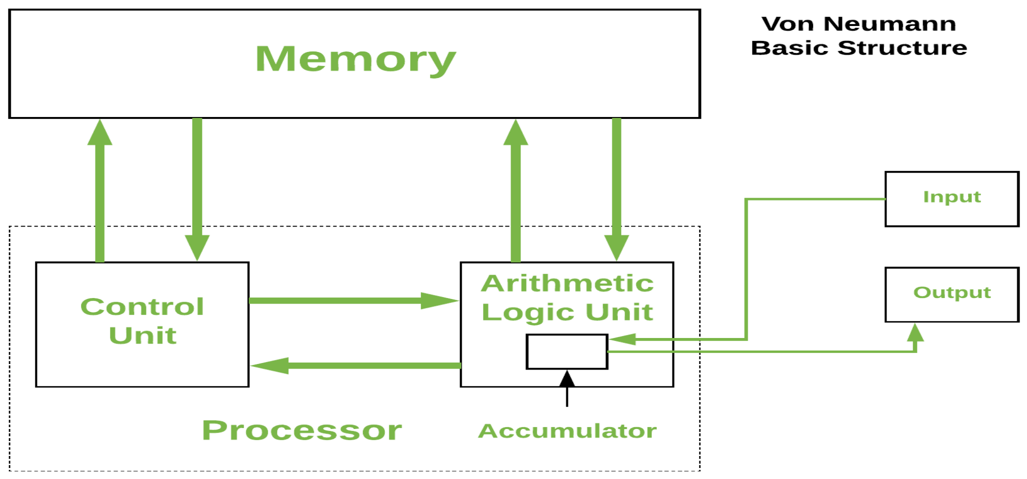
Von-Neumann Architecture:

Historically there have been 2 types of Computers:

1. **Fixed Program Computers –** Their function is very specific and they couldn't be re-programmed, e.g. Calculators.
2. **Stored Program Computers –** These can be programmed to carry out many different tasks, applications are stored on them, hence the name.

The modern computers are based on a stored-program concept introduced by John Von Neumann. In this stored-program concept, programs and data are stored in a separate storage unit called memories and are treated the same. This novel idea meant that a computer built with this architecture would be much easier to reprogram.

The basic structure is like this,



It is also known as **ISA** (Instruction set architecture) computer and is having three basic units:

1. The Central Processing Unit (CPU)
2. The Main Memory Unit
3. The Input/Output Device

Let's consider them in details.

- **Control Unit –**
  A control unit (CU) handles all processor control signals. It directs all input and output flow, fetches code for instructions, and controls how data moves around the system.
- **Arithmetic and Logic Unit (ALU) –**
  The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations, and Arithmetic operations.

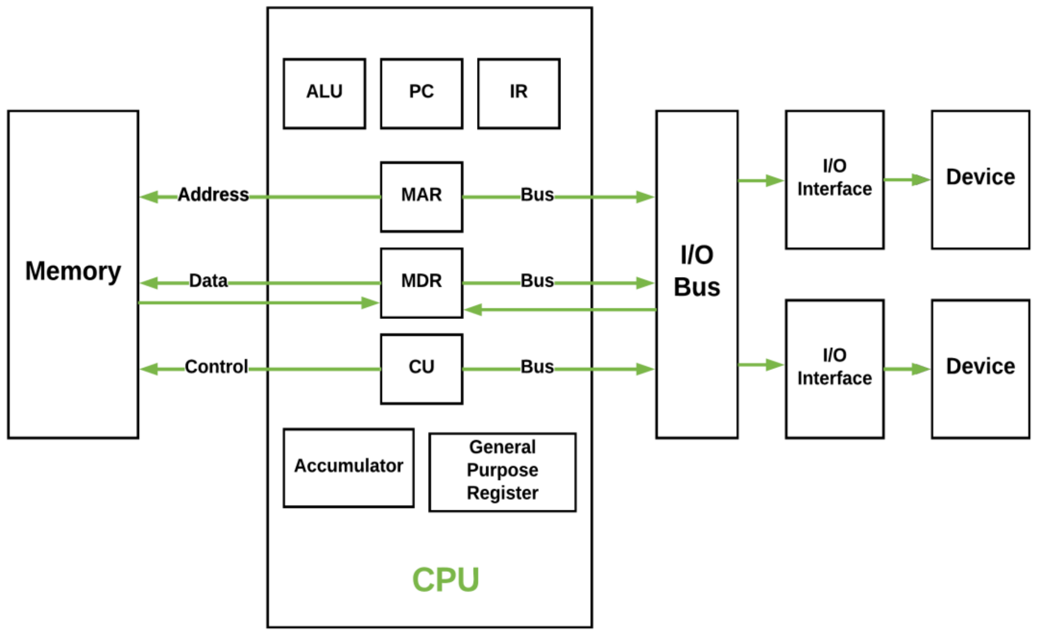


Figure – Basic CPU structure, illustrating ALU

- **Main Memory Unit (Registers) –**
  1. **Accumulator:** Stores the results of calculations made by ALU.

2. **Program Counter (PC):** Keeps track of the memory location of the next instructions to be dealt with. The PC then passes this next address to Memory Address Register (MAR).
3. **Memory Address Register (MAR):** It stores the memory locations of instructions that need to be fetched from memory or stored into memory.
4. **Memory Data Register (MDR):** It stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.
5. **Current Instruction Register (CIR):** It stores the most recently fetched instructions while it is waiting to be coded and executed.
6. **Instruction Buffer Register (IBR):** The instruction that is not to be executed immediately is placed in the instruction buffer register IBR.

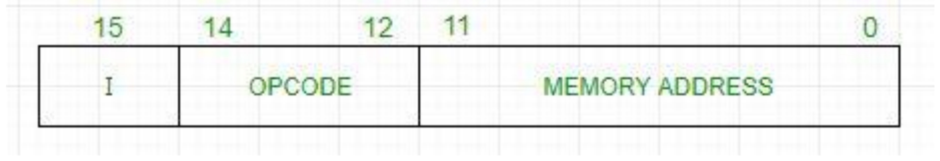
- **Input/Output Devices –** Program or data is read into main memory from the *input device* or secondary storage under the control of CPU input instruction. *Output devices* are used to output the information from a computer. If some results are evaluated by computer and it is stored in the computer, then with the help of output devices, we can present them to the user.
- **Buses –** Data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory, by the means of Buses. Types:
  1. **Data Bus:** It carries data among the memory unit, the I/O devices, and the processor.
  2. **Address Bus:** It carries the address of data (not the actual data) between memory and processor.
  3. **Control Bus:** It carries control commands from the CPU (and status signals from other devices) in order to control and coordinate all the activities within the computer.

Von Neumann bottleneck –
Whatever we do to enhance performance, we cannot get away from the fact that instructions can only be done one at a time and can only be carried out sequentially. Both of these factors hold back the competence of the CPU. This is commonly referred to as the 'Von Neumann bottleneck'. We can provide a Von Neumann processor with more cache, more RAM, or faster components but if original gains are to be made in CPU performance then an influential inspection needs to take place of CPU configuration.
This architecture is very important and is used in our PCs and even in Super Computers.

Basic Computer Instructions

The basic computer has 16-bit instruction register (IR) which can denote either memory reference or register reference or input-output instruction.

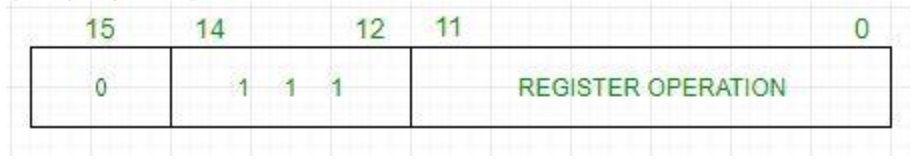
1. **Memory Reference –** These instructions refer to memory address as an operand. The other operand is always accumulator. Specifies 12-bit address, 3-bit opcode (other than 111) and 1-bit addressing mode for direct and indirect addressing.



Example –
IR register contains = 0001XXXXXXXXXXXX, i.e. ADD after fetching and decoding of instruction we find out that it is a memory reference instruction for ADD operation.
Hence, DR ← M[AR]

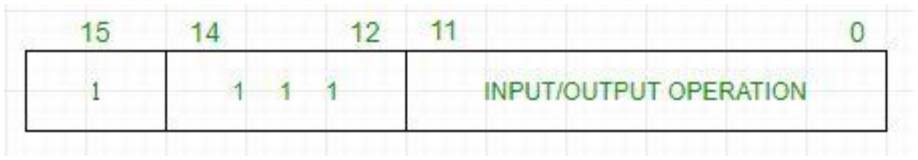
AC ← AC + DR, SC ← 0

2. **Register Reference –** These instructions perform operations on registers rather than memory addresses. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 0 (differentiates it from input/output instructions). The rest 12 bits specify register operation.



Example –
IR register contains = 0111001000000000, i.e. CMA after fetch and decode cycle we find out that it is a register reference instruction for complement accumulator.
Hence, AC ← ~AC

3. **Input/Output –** These instructions are for communication between computer and outside environment. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 1 (differentiates it from register reference instructions). The rest 12 bits specify I/O operation.



Example –
IR register contains = 1111100000000000, i.e. INP after fetch and decode cycle we find out that it is an input/output instruction for inputing character. Hence, INPUT character from peripheral device.
The set of instructions incorporated in16 bit IR register are:

1. Arithmetic, logical and shift instructions (and, add, complement, circulate left, right, etc)
2. To move information to and from memory (store the accumulator, load the accumulator)
3. Program control instructions with status conditions (branch, skip)
4. Input output instructions (input character, output character)

Symbol	Hexadecimal Code		Description
AND	0xxx	8xxx	And memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store AC content in memory
BUN	4xxx	Cxxx	Branch Unconditionally
BSA	5xxx	Dxxx	Branch and Save Return Address
ISZ	6xxx	Exxx	Increment and skip if 0
CLA	7800		Clear AC
CLE	7400		Clear E(overflow bit)
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC > 0
SNA	7008		Skip next instruction if AC < 0
SZA	7004		Skip next instruction if AC = 0

Symbol	Hexadecimal Code	Description
SZE	7002	Skip next instruction if E = 0
HLT	7001	Halt computer
INP	F800	Input character to AC
OUT	F400	Output character from AC
SKI	F200	Skip on input flag
SKO	F100	Skip on output flag
ION	F080	Interrupt On
IOF	F040	Interrupt Off

Instruction Formats (Zero, One, Two and Three Address Instruction):

A computer performs a task based on the instruction provided. Instruction in computers comprises groups called fields. These fields contain different information as for computers everything is in 0 and 1 so each field has different significance based on which a CPU decides what to perform. The most common fields are:

- Operation field specifies the operation to be performed like addition.
- Address field which contains the location of the operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

Instruction is of variable length depending upon the number of addresses it contains. Generally, CPU organization is of three types based on the number of address fields:

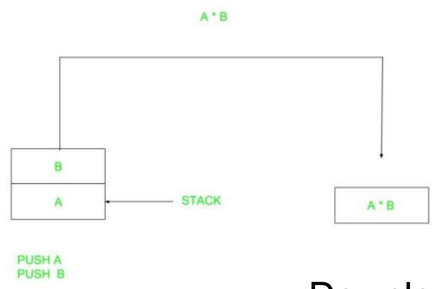
1. Single Accumulator organization
2. General register organization
3. Stack organization

In the first organization, the operation is done involving a special register called the accumulator. In second on multiple registers are used for the computation purpose. In the third organization the work on stack basis operation due to which it does not contain any address field. Only a single organization doesn't need to be applied, a blend of various organizations is mostly what we see generally.

Based on the number of address, instructions are classified as:

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

1. **Zero Address Instructions –**



A stack-based computer does not use the address field in the instruction. To evaluate an expression first it is converted to reverse Polish Notation i.e. Postfix Notation.

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

Postfixed : X = AB+CD+*

TOP means top of stack

M[X] is any memory location

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

2 .One Address Instructions –
This uses an implied ACCUMULATOR register for data manipulation. One operand is in the accumulator and the other is in the register or memory location. Implied means that the CPU already knows that one operand is in the accumulator so there is no need to specify it.

opcode	operand/address of operand	mode

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

AC is accumulator

M[] is any memory location

M[T] is temporary location

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC
LOAD	C	AC = M[C]

ADD	D	AC = AC + M[D]
MUL	T	AC = AC * M[T]
STORE	X	M[X] = AC

3.Two Address Instructions –
This is common in commercial computers. Here two addresses can be specified in the instruction. Unlike earlier in one address instruction, the result was stored in the accumulator, here the result can be stored at different locations rather than just accumulators, but require more number of bit to represent address.

opcode	Destination address	Source address	mode

Here destination address can also contain operand.

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

R1, R2 are registers

M[] is any memory location

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

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

opcode	Destination address	Source address	Source address	mode

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

R1, R2 are registers

M[] is any memory location

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

INSTRUCTION SET ARCHITECTURE

In this article, we look at what an *Instruction Set Architecture (ISA)* is and what is the difference between an **'ISA'** and *Microarchitecture*. An **ISA** is defined as the design of a computer from the *Programmer's Perspective*.
This basically means that an **ISA** describes the **design of a Computer** in terms of the **basic operations** it must support. The ISA is not concerned with the implementation-specific details of a computer. It is only concerned with the set or collection of basic operations the computer must support. For example, the AMD Athlon and the Core 2 Duo processors have entirely different implementations but they support more or less the same set of basic operations as defined in the x86 Instruction Set.
Let us try to understand the Objectives of an ISA by taking the example of the **MIPS ISA**. MIPS is one of the most widely used ISAs in education due to its simplicity.

1. The ISA defines the **types of instructions** to be supported by the processor.
   Based on the type of operations they perform MIPS Instructions are classified into 3 types:
   - **Arithmetic/Logic Instructions:**
     These Instructions perform various Arithmetic & Logical operations on one or more operands.
   - **Data Transfer Instructions:**
     These instructions are responsible for the transfer of instructions from memory to the processor registers and vice versa.
   - **Branch and Jump Instructions:**
     These instructions are responsible for breaking the sequential flow of instructions and jumping to instructions at various other locations, this is necessary for the implementation of *functions* and *conditional statements*.

2. The ISA defines the **maximum length** of each type of instruction.
   Since the MIPS is a 32 bit ISA, each instruction must be accommodated within 32 bits.

3. The ISA defines the **Instruction Format** of each type of instruction.
   The **Instruction Format** determines how the entire instruction is encoded within 32 bits
   There are 3 types of Instruction Formats in the MIPS ISA:
   - R-Instruction Format
   - I-Instruction Format
   - J-Instruction Format

If we look at the Abstraction Hierarchy:

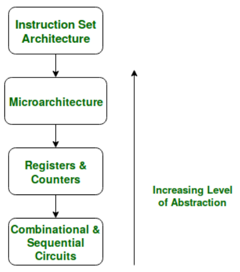

Instruction Set Architecture
Microarchitecture
Registers & Counters
Combinational & Sequential Circuits
Increasing Level of Abstraction

Figure – The Abstraction Hierarchy
We note that the **Microarchitectural** level lies just below the **ISA** level and hence is concerned with the implementation of the basic operations to be supported by the Computer as defined by the **ISA**. Therefore we can say that the AMD Athlon and Core 2 Duo processors are based on the same ISA but have different microarchitectures with different performance and efficiencies.
Now one may ask the need to distinguish between **Microarchitecture** and **ISA**?
The answer to this lies in the need to standardize and maintain the compatibility of programs across different hardware implementations based on the same **ISA**. Making different machines compatible with the same set of basic instructions (The ISA) allows the same program to run smoothly on many different machines thereby making it easier for the programmers to document and maintain code for many different machines simultaneously and efficiently.
This Flexibility is the reason we first define an ISA and then design different microarchitectures complying with this ISA for implementing the machine. The design of a lower-level ISA is one of the major tasks in the study of Computer Architecture.

Instruction Set Architecture	Microarchitecture

Instruction Set Architecture	Microarchitecture
The ISA is responsible for defining the set of instructions to be supported by the processor. For example, some of the instructions defined by the ARMv7 ISA are given below.	The Microarchitecture is more concerned with the lower level implementation of how the instructions are going to be executed and deals with concepts like Instruction Pipelining, Branch Prediction, Out of Order Execution.
The Branch of **Computer Architecture** is more inclined towards the Analysis and Design of Instruction Set Architecture. For Example, Intel developed the *x86* architecture, ARM developed the *ARM* architecture, & AMD developed the *amd64* architecture. The RISC-V ISA developed by UC Berkeley is an example of an Open Source ISA.	On the other hand, the Branch of **Computer Organization** is concerned with the implementation of a particular ISA deals with various hardware implementation techniques, i.e. is the Microarchitecture level. For Example, ARM licenses other companies like Qualcomm, Apple for using ARM ISA, but each of these companies have their own implementations of this ISA thereby making them different in performance and power efficiency. The *Krait* cores developed by Qualcomm have a different microarchitecture and the Apple A-series processors have a different microarchitecture.

The *x86* was developed by Intel, but we see that almost every year Intel comes up with a new generation of i-series processors. The *x86* architecture on which most of the Intel Processors are based essentially remains the same across all these generations but, where they differ is in the underlying Microarchitecture. They differ in their implementation and hence are claimed to have improved Performance. These various Microarchitectures developed by Intel are codenamed as 'Nehalem', 'Sandybridge', 'Ivybridge', and so on.
Therefore, in conclusion, we can say that different machines may be based on the same ISA but have different Microarchitectur es.

INSTRUCTIONS & INSTRUCTION SEQUENCING

The tasks carried out by a computer program consist of a sequence of small steps, suchas adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen.

A computer must have instructions capable of performing 4 types of operations:

1) Data transfers between the memory and the registers (MOV, PUSH, POP, XCHG).

2) Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT).

3) Program sequencing and control(CALL.RET, LOOP, INT).

4) I/0 transfers (IN, OUT).

REGISTER TRANSFER NOTATION (RTN)

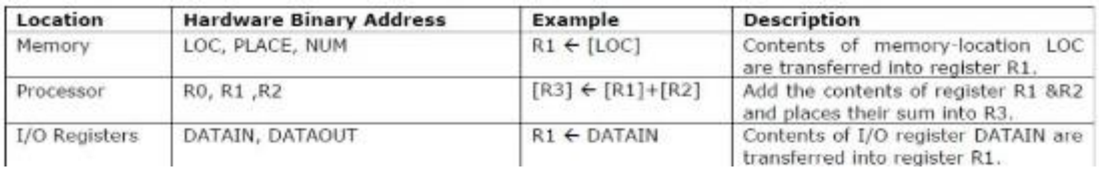
Here we describe the transfer of information from one location in a computer to another.Possible locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O subsystem. Most of the time, we identify such locationssymbolically with convenient names.

• The possible locations in which transfer of information occurs are:

1) Memory-location

2) Processor register &

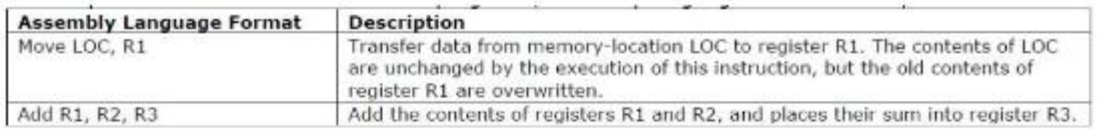
3) Registers in I/O device.



Location	Hardware Binary Address	Example	Description
Memory	LOC, PLACE, NUM	R1 ← [LOC]	Contents of memory-location LOC are transferred into register R1.
Processor	R0, R1 ,R2	[R3] ← [R1]+[R2]	Add the contents of register R1 &R2 and places their sum into R3.
I/O Registers	DATAIN, DATAOUT	R1 ← DATAIN	Contents of I/O register DATAIN are transferred into register R1.

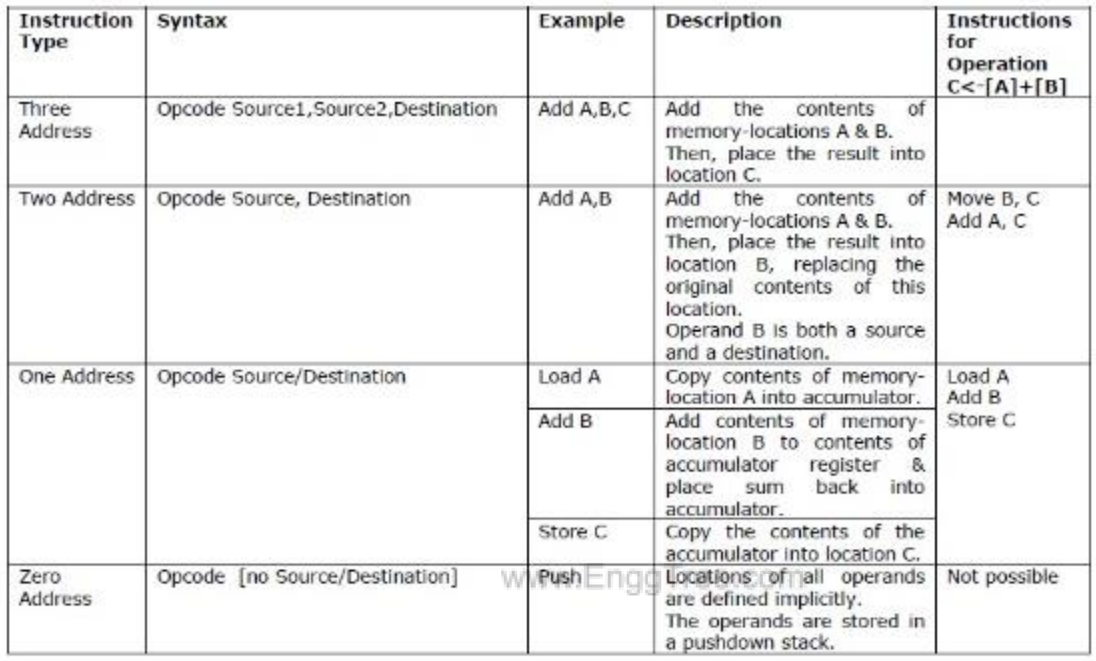
ASSEMBLY LANGUAGE NOTATION

• To represent machine instructions and programs, assembly language format is used.



Assembly Language Format	Description
Move LOC, R1	Transfer data from memory-location LOC to register R1. The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.
Add R1, R2, R3	Add the contents of registers R1 and R2, and places their sum into register R3.

BASIC INSTRUCTION TYPES



Instruction Type	Syntax	Example	Description	Instructions for Operation C<-[A]+[B]
Three Address	Opcode Source1,Source2,Destination	Add A,B,C	Add the contents of memory-locations A & B. Then, place the result into location C.	
Two Address	Opcode Source, Destination	Add A,B	Add the contents of memory-locations A & B. Then, place the result into location B, replacing the original contents of this location. Operand B is both a source and a destination.	Move B, C Add A, C
One Address	Opcode Source/Destination	Load A	Copy contents of memory-location A into accumulator.	Load A Add B Store C
		Add B	Add contents of memory-location B to contents of accumulator register & place sum back into accumulator.	
		Store C	Copy the contents of the accumulator into location C.	
Zero Address	Opcode [no Source/Destination]	Push	Locations of all operands are defined implicitly. The operands are stored in a pushdown stack.	Not possible

INSTRUCTION EXECUTION & STRAIGHT LINE SEQUENCING

• The program is executed as follows:

1) Initially, the address of the first instruction is loaded into PC (Figure 2.8).

2) Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called Straight-Line sequencing.

3) During the execution of each instruction, PC is incremented by 4 to point to next instruction.

• There are 2 phases for Instruction Execution:

1) Fetch Phase: The instruction is fetched from the memory-location and placed in the IR.

2) Execute Phase: The contents of IR is examined to determine which operation is to beperformed. The specified-operation is then performed by the processor.

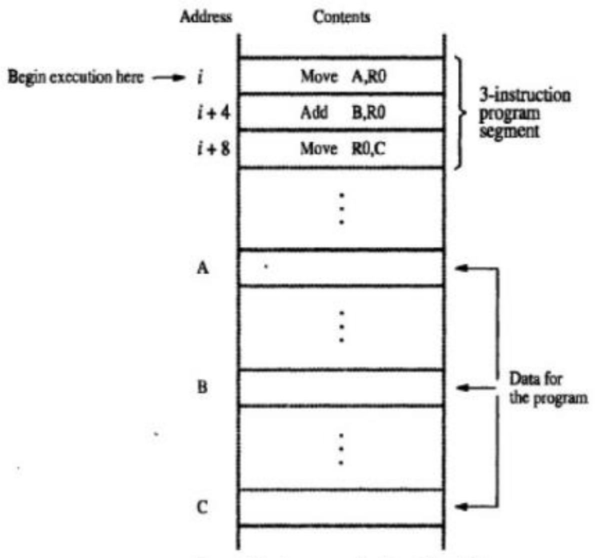


Figure 2.8 A program for C ← [A] + [B].

Program Explanation

• Consider the program for adding a list of n numbers (Figure 2.9).

• The Address of the memory-locations containing the n numbers are symbolically given as NUM1, NUM2…..NUMn.

• Separate Add instruction is used to add each number to the contents of register R0.

• After all the numbers have been added, the result is placed in memory-location SUM.

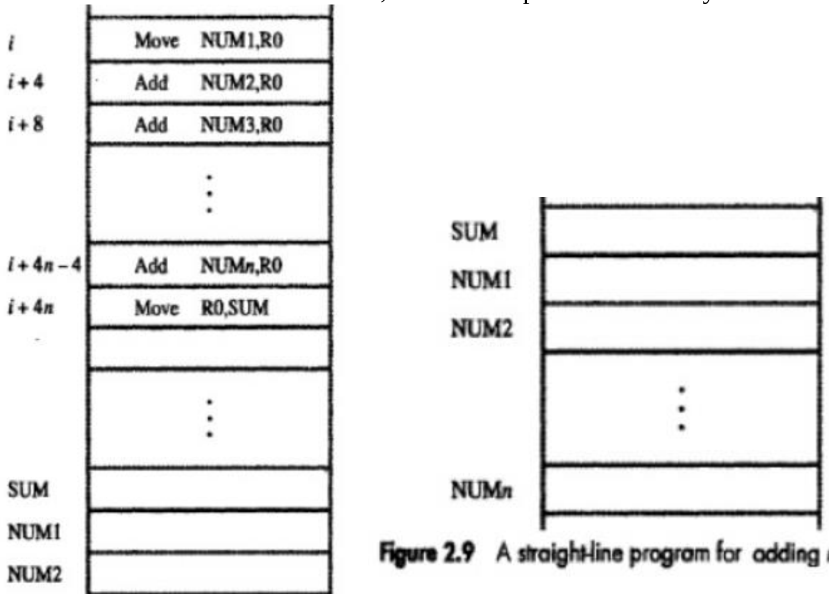


Figure 2.9 A straight-line program for adding n numbers.

ADDRESSING MODES:

The different ways in which the location of an operand is specified in an instruction are referred to as

1)Immediate Mode

• The operand is given explicitly in the instruction.

• For example, the instruction Move #200, R0 ;Place the value 200 in register R0.

• Clearly, the immediate mode is only used to specify the value of a source-operand.

2)Register Mode

• The operand is the contents of a register.

• The name (or address) of the register is given in the instruction.

• Registers are used as temporary storage locations where the data in a register are accessed.

• For example, the instruction Move R1, R2 ;Copy content of register R1 into register R2.

3)Absolute (Direct) Mode

• The operand is in a memory-location.

• The address of memory-location is given explicitly in the instruction.

• The absolute mode can represent global variables in the program.

• For example, the instruction Move LOC, R2 ;Copy content of memory-location LOC into register R2.

4)INDIRECTION AND POINTERS

• Instruction does not give the operand or its address explicitly.

• Instead, the instruction provides information from which the new address of the operand can be determined.

• This address is called Effective Address (EA) of the operand.

Indirect Mode

• The EA of the operand is the contents of a register(or memory-location).
• The register (or memory-location) that contains the address of an operand is called a Pointer.
• We denote the indirection by → name of the register or → new address given in the instruction. E.g: Add (R1), R0;The operand is in memory. Register R1 gives the effective-address (B) of the operand. The data is read from location B and added to contents of register R0.

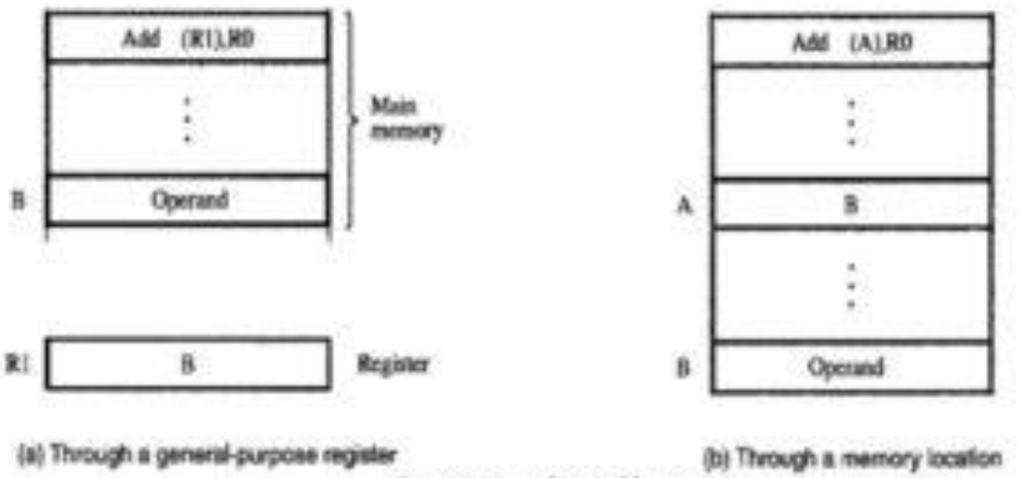
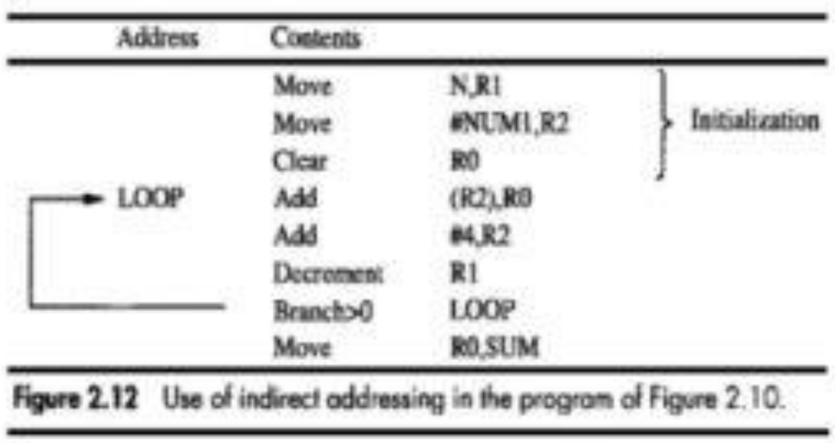

Add (R1),R0
Main memory
B
Operand
R1
B
Register
(a) Through a general-purpose register
Add (A),R0
A
B
B
Operand
(b) Through a memory location

Figure 2.11 Indirect addressing

• To execute the Add instruction in fig 2.11 (a), the processor uses the value which is in register R1, as the EA of the operand.

• It requests a read operation from the memory to read the contents of location B. The value read is the desired operand, which the processor adds to the contents of register R0.

• Indirect addressing through a memory-location is also possible as shown in fig 2.11(b). In this case, the processor first reads the contents of memory-location A, then requests a second read operation using the value B as an address to obtain the operand.



```
Address     Contents

            Move        N,R1        }
            Move        #NUM1,R2    } Initialization
            Clear       R0          }
LOOP        Add         (R2),R0
            Add         #4,R2
            Decrement   R1
            Branch>0    LOOP
            Move        R0,SUM
```

Figure 2.12 Use of indirect addressing in the program of Figure 2.10.

Program Explanation

• In above program, Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2.

• The initialization-section of the program loads the counter-value n from memory-location N into R1 and uses the immediate addressing-mode to place the address value NUM1, which is the address of the first number in the list, into R2. Then it clears R0 to 0.

• The first two instructions in the loop implement the unspecified instruction block starting at LOOP.

• The first time through the loop, the instruction Add (R2), R0 fetches the operand at location NUM1 and adds it to R0.

• The second Add instruction adds 4 to the contents of the pointer R2, so that it will contain the address value NUM2 when the above instruction is executed in the second pass through the loop.

5)INDEXING AND ARRAYS

• A different kind of flexibility for accessing operands is useful in dealing with lists and arrays. Index mode

• The operation is indicated as X(Ri) where X=the constant value which defines an offset(also called a displacement). Ri=the name of the index register which contains address of a new location.

• The effective-address of the operand is given by EA=X+[Ri]

• The contents of the index-register are not changed in the process of generating the effectiveaddress.

• The constant X may be given either → as an explicit number or → as a symbolic-name representing a numerical value.

6)Base with Index Mode

• Another version of the Index mode uses 2 registers which can be denoted as(Ri, Rj)

• Here, a second register may be used to contain the offset X.

• The second register is usually called the base register.

• The effective-address of the operand is given by EA=[Ri]+[Rj]

• This form of indexed addressing provides more flexibility in accessing operands because both components of the effective-address can be changed.

7)Base with Index & Offset Mode

• Another version of the Index mode uses 2 registers plus a constant, which can be denoted as X(Ri, Rj)

• The effective-address of the operand is given by EA=X+[Ri]+[Rj]

•This added flexibility is useful in accessing multiple components inside each item in a record, where the beginning of an item is specified by the (Ri, Rj) part of the addressing-mode. In other words, this mode implements a 3-dimensional array.

8)RELATIVE MODE

• This is similar to index-mode with one difference: The effective-address is determined using the PC in place of the general purpose register Ri.

• The operation is indicated as X(PC).

• X(PC) denotes an effective-address of the operand which is X locations above or below the current contents of PC.

• Since the addressed-location is identified "relative" to the PC, the name Relative mode is associated with this type of addressing.

• This mode is used commonly in conditional branch instructions.

• An instruction such as Branch > 0 LOOP ;Causes program execution to go to the branch target locationidentified by name LOOP if branch condition is satisfied.

9) Auto Increment Mode

Effective-address of operand is contents of a register specified in the instruction (Fig: 2.16). After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list. Implicitly, the increment amount is 1. This mode is denoted as (Ri)+ ;whereRi=pointer-register.

10) Auto Decrement Mode The contents of a register specified in the instruction are first automatically decremented and are then used as the effective-address of the operand. This mode is denoted as -(Ri) ;where Ri=pointer-register. These 2 modes can be used together to implement an important data structure called a stack.

ENCODING OF MACHINE INSTRUCTIONS

• To be executed in a processor, an instruction must be encoded in a binary-pattern. Such encoded instructions are referred to as Machine Instructions.

• The instructions that use symbolic-names and acronyms are called assembly language instructions.

• We have seen instructions that perform operations such as add, subtract, move, shift, rotate, and branch. These instructions may use operands of different sizes, such as 32-bit and 8-bit numbers.

• Let us examine some typical cases.

The instruction Add R1, R2 ; Has to specify the registers R1 and R2, in addition to the OP code. If the processor has 16 registers, then four bits are needed to identify each register. Additional bits are needed to indicate that the Register addressing-mode is used for each operand.

The instruction Move 24(R0), R5 ;Requires 16 bits to denote the OP code and the two registers, and some bits to express that the source operand uses the Index addressing mode and that the index value is 24.

• In all these examples, the instructions can be encoded in a 32-bit word (Fig 2.39).

• The OP code for given instruction refers to type of operation that is to be performed.

• Source and destination field refers to source and destination operand respectively.

• The "Other info" field allows us to specify the additional information that may be needed such as an index value or an immediate operand.

• Using multiple words, we can implement complex instructions, closely resembling operations in high level programming languages. The term complex instruction set computers (CISC) refers to processors that use

• CISC approach results in instructions of variable length, dependent on the number of operands and the type of addressing modes used.

• In RISC (reduced instruction set computers), any instruction occupies only one word.

• The RISC approach introduced other restrictions such as that all manipulation of data must be done on operands that are already in registers. Ex: Add R1,R2,R3

• In RISC type machine, the memory references are limited to only Load/Store operations.Ro

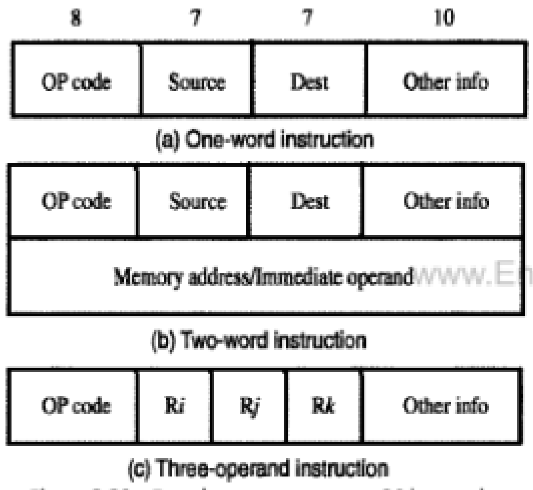


Figure 2.39 Encoding instructions into 32-bit words.

Rotate operations:

• In shift operations, the bits shifted out of the operand are lost, except for the last bit shifted out which is retained in the Carry-flag C.

• To preserve all bits, a set of rotate instructions can be used.

• They move the bits that are shifted out of one end of the operand back into the other end.

• Two versions of both the left and right rotate instructions are usually provided. In one version, the bits of the operand is simply rotated. In the other version, the rotation includes the C flag.

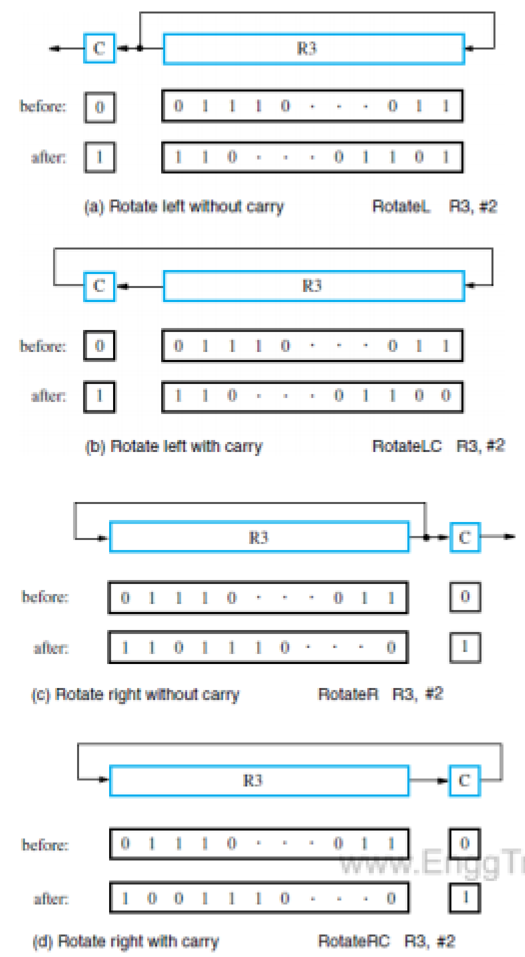


Figure 2.25 Rotate instructions.

SHIFT AND ROTATE INSTRUCTIONS

• There are many applications that require the bits of an operand to be shifted right or left some specified number of bit positions.

• The details of how the shifts are performed depend on whether the operand is a signed number or some more general binary-coded information.

• For general operands, we use a logical shift. For a number, we use an arithmetic shift, which preserves the sign of the number.

LOGICAL SHIFTS

• Two logical shift instructions are

1) Shifting left (LShiftL) &

2) Shifting right (LShiftR).

• These instructions shift an operand over a number of bit positions specified in a count operand contained in the instruction.

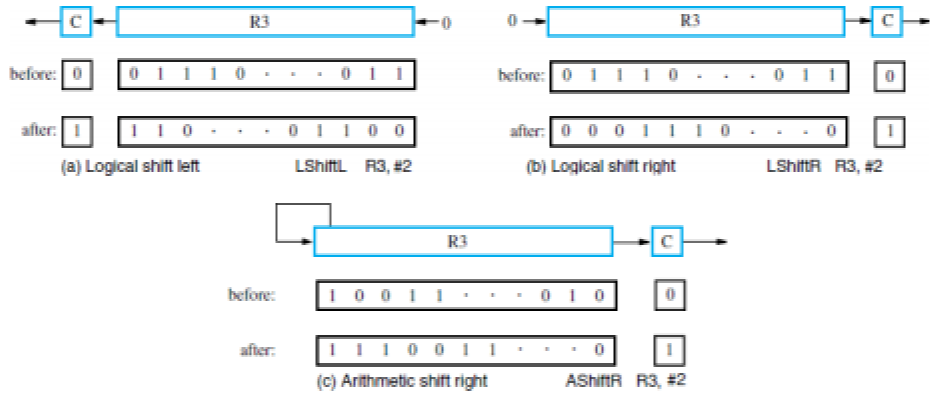
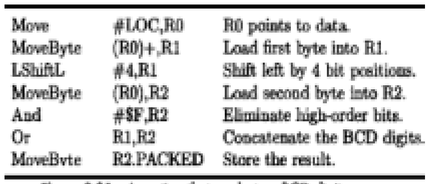

C
R3
0
before: 0 | 0 1 1 1 0 · · · 0 1 1
after: 1 | 1 1 0 · · · 0 1 1 0 0
(a) Logical shift left LShiftL R3, #2
0
R3
C
before: 0 1 1 1 0 · · · 0 1 1 | 0
after: 0 0 0 1 1 1 0 · · · 0 | 1
(b) Logical shift right LShiftR R3, #2
R3
C
before: 1 0 0 1 1 · · · 0 1 0 | 0
after: 1 1 1 0 0 1 1 · · · 0 | 1
(c) Arithmetic shift right AShiftR R3, #2

Figure 2.23 Logical and arithmetic shift instructions.



```
Move       #LOC,R0     R0 points to data.
MoveByte   (R0)+,R1    Load first byte into R1.
LShiftL    #4,R1       Shift left by 4 bit positions.
MoveByte   (R0),R2     Load second byte into R2.
And        #$F,R2      Eliminate high-order bits.
Or         R1,R2       Concatenate the BCD digits.
MoveByte   R2,PACKED   Store the result.
```

Figure 2.31 A routine that packs two BCD digits.

Interaction between Assembly language and high levellanguage

High level languages such as Python, C, C++, Java, C# allows the programmer towrite the alpha numeric codes.

The compiler converts the alpha numeric codes into assembly code.Assembly

language codes example:

ADD R1, R2, R3

ADDi R1, R2, 20

LOAD R1, 20(R2)

STORE R1, 20(R2)

The assembler intern converts the assembly language code into binary code(machine code).

Difference between assembly language and high level language

Parameters	**Assembly Language**	**High-Level Language**
Conversion	The assembly language requires an assembler for the process of conversion.	A high-level language requires an interpreter/ compiler for the process of conversion.
Process of Conversion	We perform the conversion of an assembly language into a machine language.	We perform the conversion of a high-level language into an assembly language and then into a machine-level language for the computer.
Machine Dependency	The assembly language is a machine-dependent type of language.	A high-level language is a machine-independent type of language.
Codes	It makes use of the mnemonic codes for operation.	It makes use of the English statements for operation.
Operation of Lower Level	It provides support for various low-level operations.	It does not provide any support for low-level languages.
Access to Hardware Component	Accessing the hardware component is very easy in this case.	Accessing the hardware component is very difficult in this case.
Compactness in Code	The code is more compact in this case.	No code compactness is present in this case.
Type of Processor	The program that we write for one processor in an assembly language will not run on any other processor type. It means that it is processor-dependent.	This language is processor-independent. It means that the programs that we write using high-level languages can easily run on any processor independent of its type.
Accuracy	It has better accuracy.	Accuracy is much lesser in this case.
Performance	An assembly language performs better than any high-level language, in general.	The performance is comparatively not so good.

Length of Executable Code	It is shorter in assembly language.	It is larger in a high-level language.
Time Taken in Code Execution	Execution of code takes less time in this case because the code is not very large.	It takes up more time for execution because it needs to execute a large code.
Efficiency	It is way more efficient because of the shorter executable codes.	It is comparatively less efficient because the executable codes are comparatively longer in length.
Reading of Pointers	We can do that directly at a physical address in the case of an assembly language.	It is not possible to do so in the case of a high-level language.
Extra Instructions	We don‟t need that in the case of an assembly language.	This language must give some extra instructions for running any code on the computer.
Ease of Understanding	It is very difficult to debug and understand the code of an assembly language.	It is very easy to debug and understand the code of an assembly language.

UNIT –IV PROCESSOR

INSTRUCTION EXECUTION

Steps in Instruction Execution by CPU:

Six steps are involved in execution of an instruction by CPU. However, not all of them are required for all instructions.

1. Fetch instruction
2. Decode information
3. Perform ALU operation
4. Access memory
5. Update register file
6. Update the Program Counter (PC)

Step 1: Fetch instruction

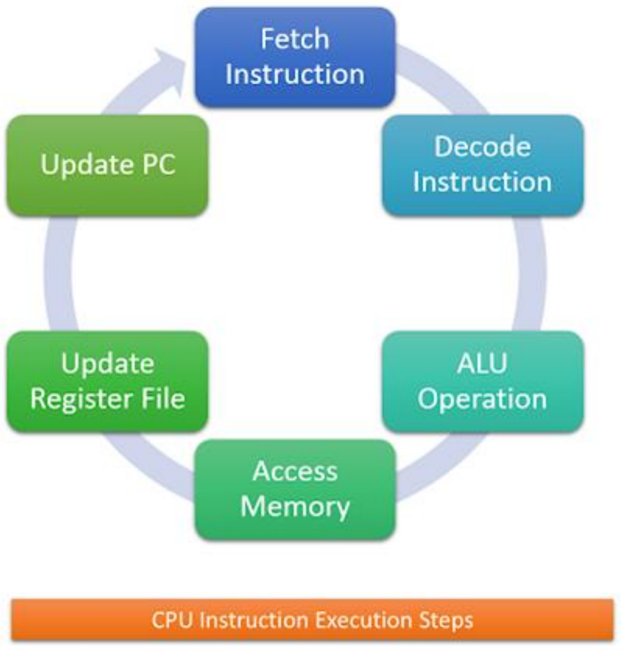
Execution cycle starts with fetching instruction from main memory. The instruction at the current program counter (PC) will be fetched and will be stored in instruction register (IR).

Step 2: Decode instruction

During this cycle the encoded instruction present in the IR (instruction register) is interpreted by the decoder.

Step 3: Perform ALU operation

ALU (Arithmetic Logic Unit) is where two operands in the instruction will be operated on given operator in the instructions. Such as, if the instruction was to add two numbers, then here the addition will happen. ALU take two values and output one, the result of the operation.



Step 4: Access memory

There are only two kind of instructions that access memory: LOAD and STORE. LOAD copies a value from memory to a register and STORE copies a register value to memory. Any other instruction skips this step.

Step 5: Update Register File

In this step, the output/result of the ALU is written back to the register file to update the register file. The result could also be due to a LOAD from memory. Some instructions don't have results to store. For example, BRANCH and JUMP instructions do not have any results to store.

Step 6: Update the PC (Program Counter)

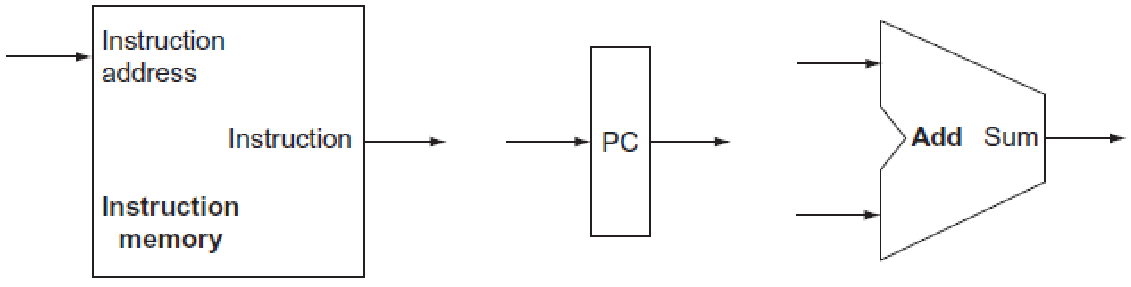
Ultimately, at the end of the execution of the current instruction, we need to update the program counter (PC) to the address of the next instruction, so that we can go back to step 1 where the CPU will fetch instruction. However, the program counter might need to be set to other memory address than the next one if the instruction was BRANCH or JUMP

BUILDING A DATAPATH

Datapath:

It is a processor component that are used to perform arithmetic operations and elements that holds the data.

Main elements of data path are Instruction memory, Program Counter (PC), ALU adder.



a. Instruction memory b. Program counter c. Adder

Building a data path

MIPS processor data path can be built incrementally by only considering the subsets of instruction.

Types of Elements in the Datapath

State element:

- A memory element, i.e., it contains a state
- E.g., program counter, instruction memory Combinational element:
- Elements that operate on values
- Eg adder ALU E.g. adder, ALU

Elements required by the different classes of instructions

- Arithmetic and logical instructions
- Data transfer instructions
- Branch instructions

R-Format ALU Instructions

- E.g., add $t1, $t2, $t3
- Perform arithmetic/logical operation
- Read two register operands and write register result

Register file:

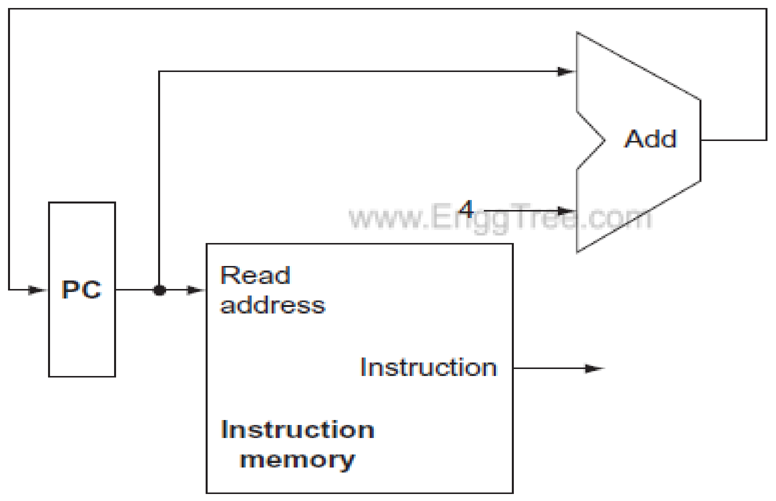
- A collection of the registers
- Any register can be read or written by specifying the number of the register
- Contains the register state of the computer

ALU

- Takes two 32 bit input and produces a 32 bit output
- Also, sets one-bit signal if the results is 0

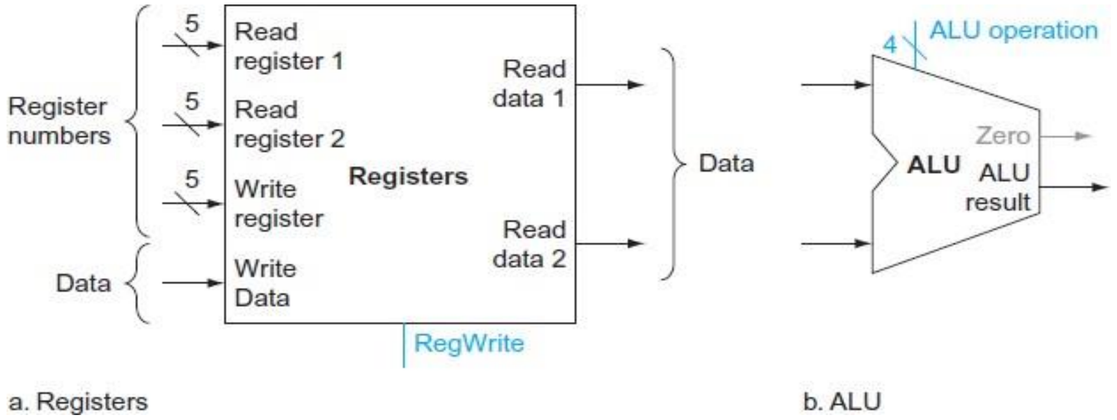
- The operation done by ALU is controlled by a 4 bit control signal input. This is set according to the instruction .

Portion of data path used for fetching instructions and incrementing the program counter



Register type instructions

ADD $t1, $t2, $t3



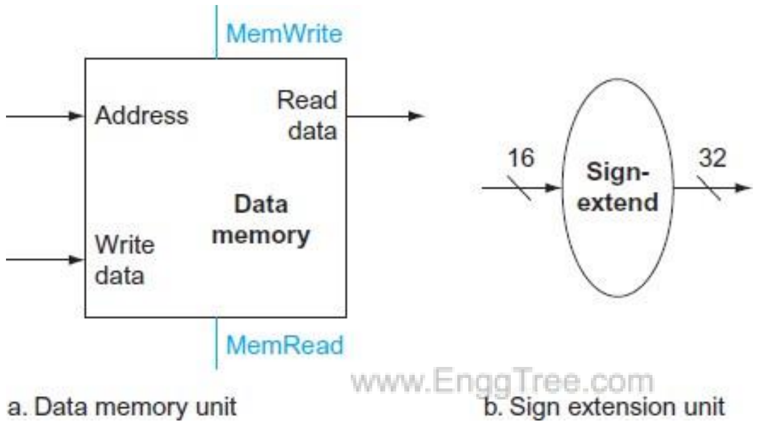
Use of sign extension unit

It is used to convert the 16 bit constant input into 32 bit constant output.

Addi $R1, $R2, 20

Here 20 is the 16 bit constant.

Sign extension unit is used to convert this 16 bit constant into 32 bit output and given to the ALU unit as a input



a. Data memory unit b. Sign extension unit

Use of shift left 2

Shift left 2 operation is used in conditional branch instruction

Beq $t0, $t1, 250

It is used to find the exact byte address from the branch address specified in word (here 250 is word no.)

Control branch : PC = PC + 250

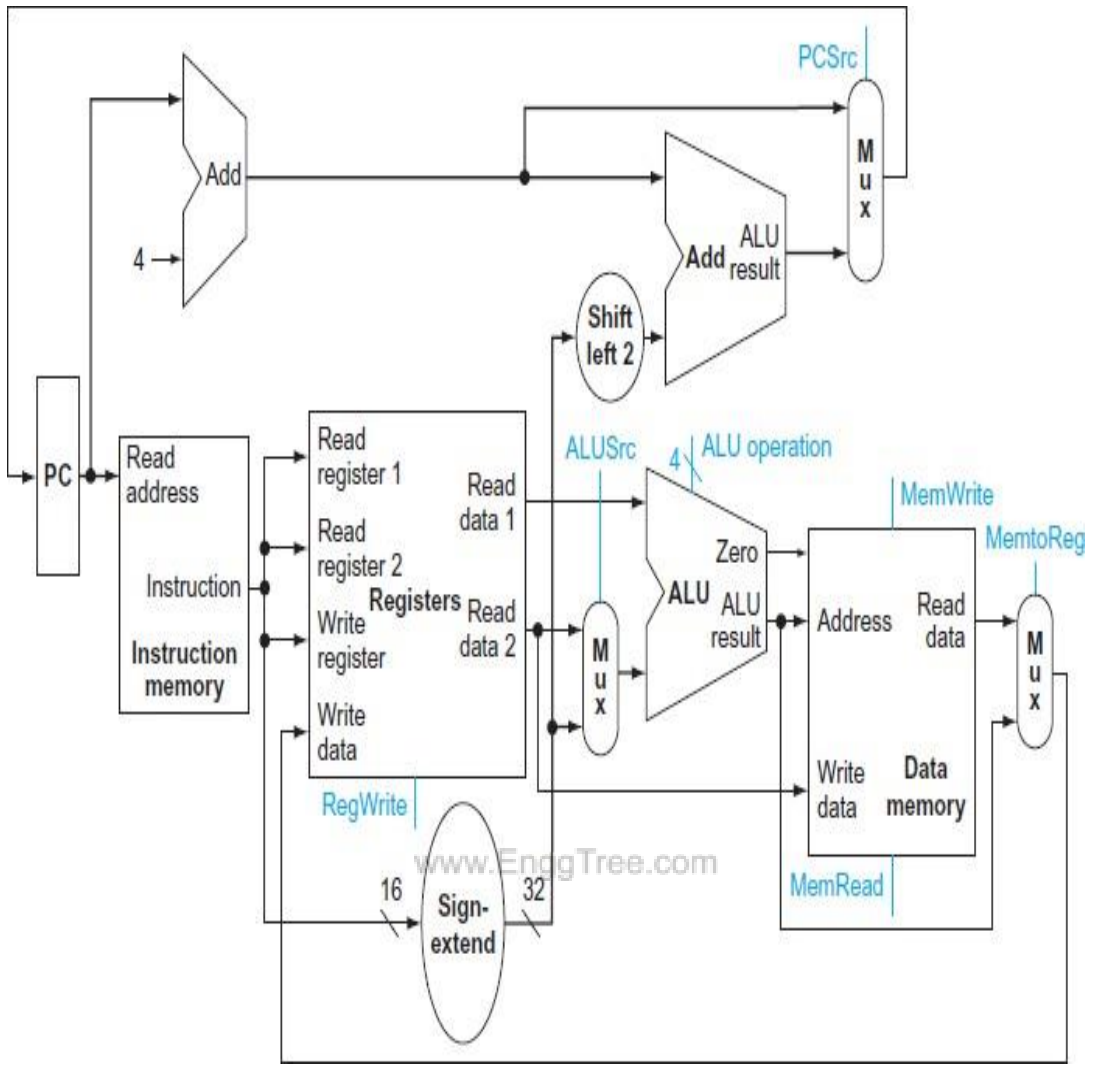
If we perform the shift left 2 operation on 250 we can obtain 1000.

The control branch address is, PC = PC +1000

<u>Creating a single data path</u>

Here all the operations are included in single data path.

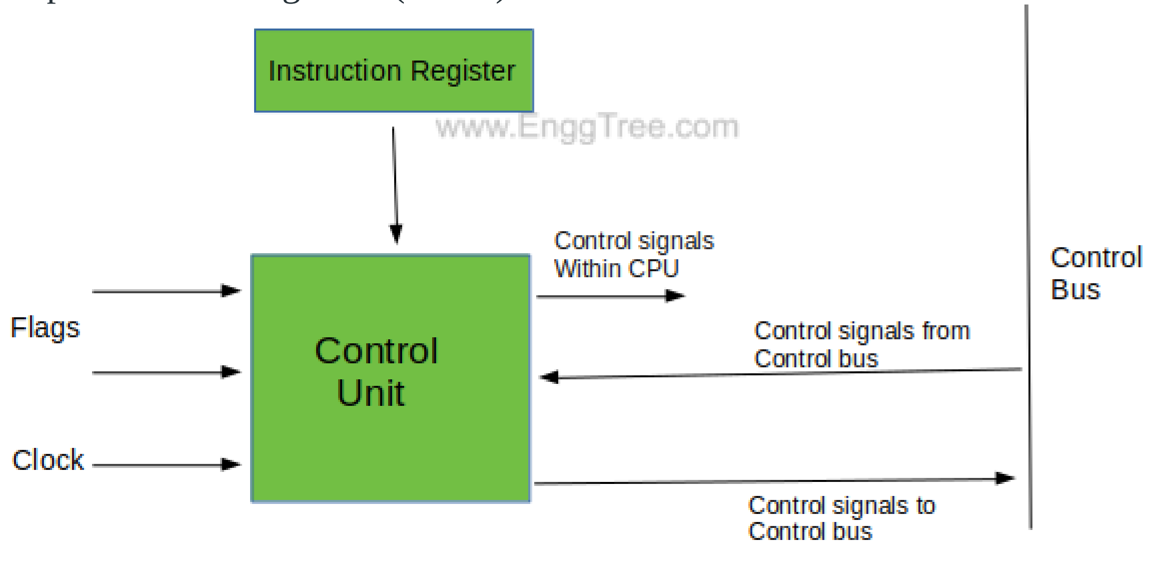
Operations such as R-Type, I-Type, J-type are included in this datapath.


PCSrc
Add
4
M u x
Add
ALU result
Shift left 2
PC
Read address
Instruction
Instruction memory
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
ALUSrc
4
ALU operation
M u x
Zero
ALU
ALU result
MemWrite
MemtoReg
Address
Read data
Write data
Data memory
MemRead
M u x
16
Sign-extend
32

DESIGNING OF CONTROL UNIT

Control Unit is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John von Neumann. It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.
A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer. Examples of devices that require a CU are:

- Control Processing Units(CPUs)
- Graphics Processing Units(GPUs)


Instruction Register
www.EnggTree.com
Control signals Within CPU
Control Bus
Flags
Control Unit
Control signals from Control bus
Clock
Control signals to Control bus

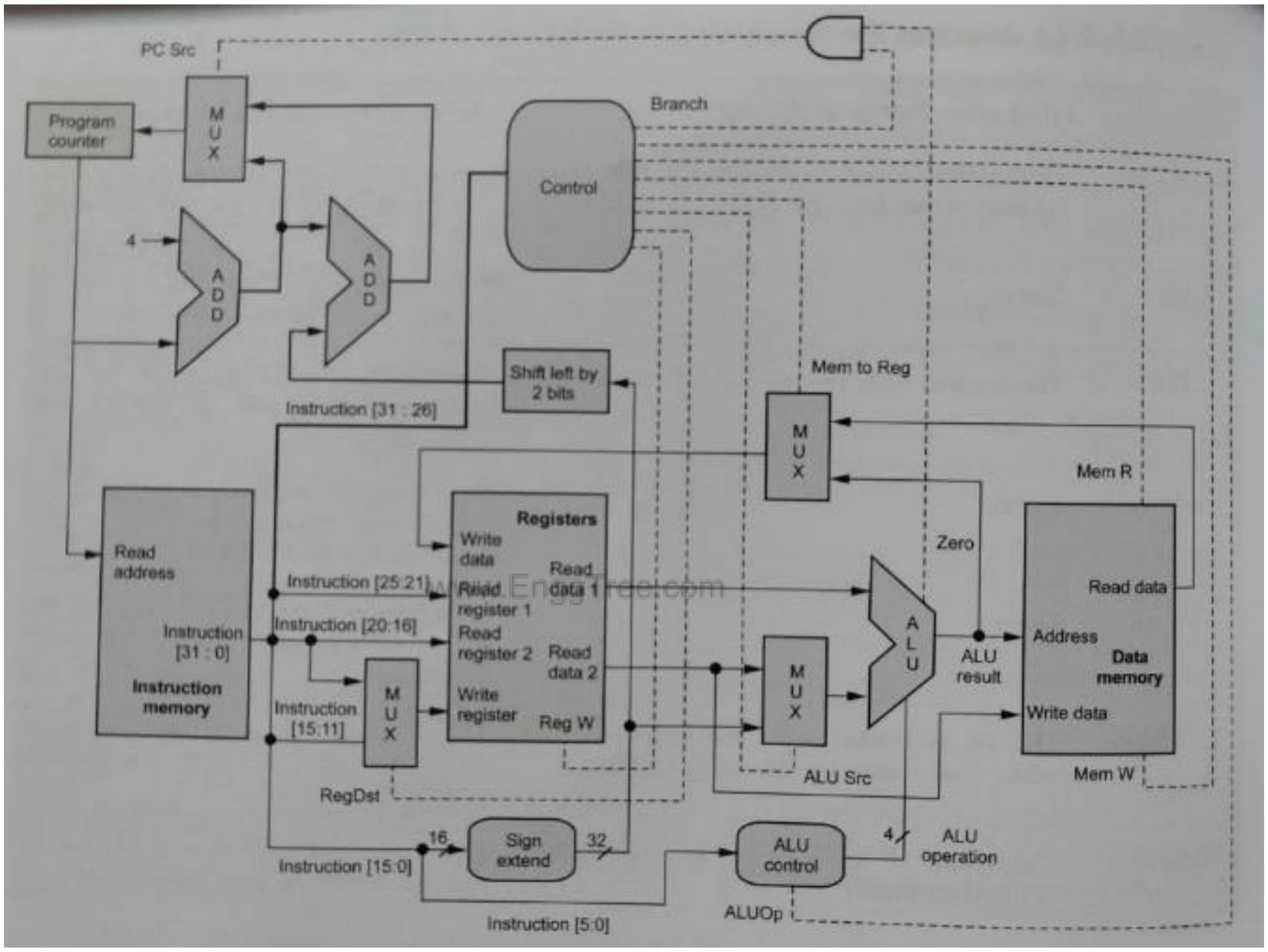
Block Diagram of the Control Unit

Functions of the Control Unit –

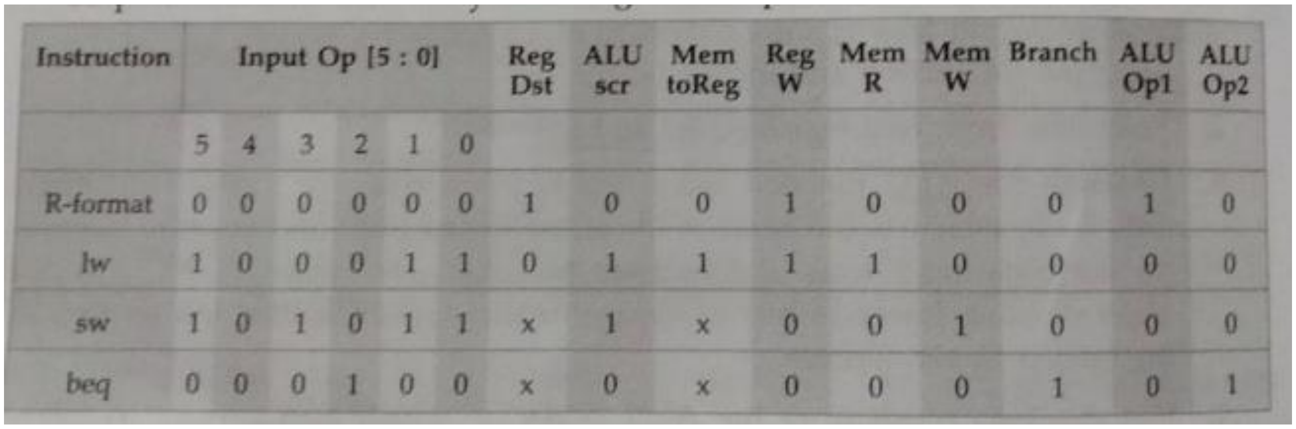
1. It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
2. It interprets instructions.
3. It controls data flow inside the processor.
4. It receives external instructions or commands to which it converts to sequence of control signals.

5. It controls many execution units(i.e. ALU, data buffers and registers) contained within a CPU.
6. It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.

Simple datapath with the control unit



Truth table for ouptut control lines and ALUOp



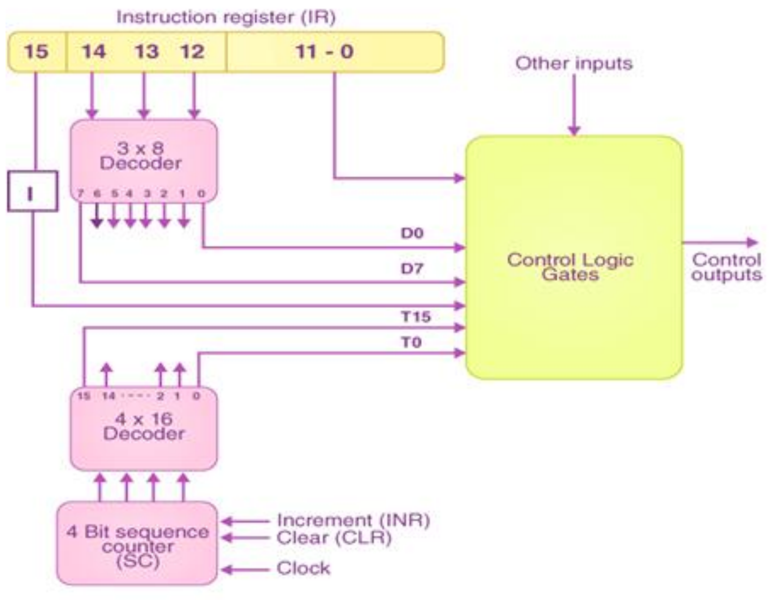
Instruction	Input Op [5 : 0]						Reg Dst	ALU scr	Mem toReg	Reg W	Mem R	Mem W	Branch	ALU Op1	ALU Op2
	5	4	3	2	1	0									
R-format	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0
lw	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0
sw	1	0	1	0	1	1	x	1	x	0	0	1	0	0	0
beq	0	0	0	1	0	0	x	0	x	0	0	0	1	0	1

Types of Control Unit –
There are two types of control units: Hardwired control unit and Microprogrammable control unit.

HARDWIRED CONTROL METHOD

It is a hardware based method to design the control unit.

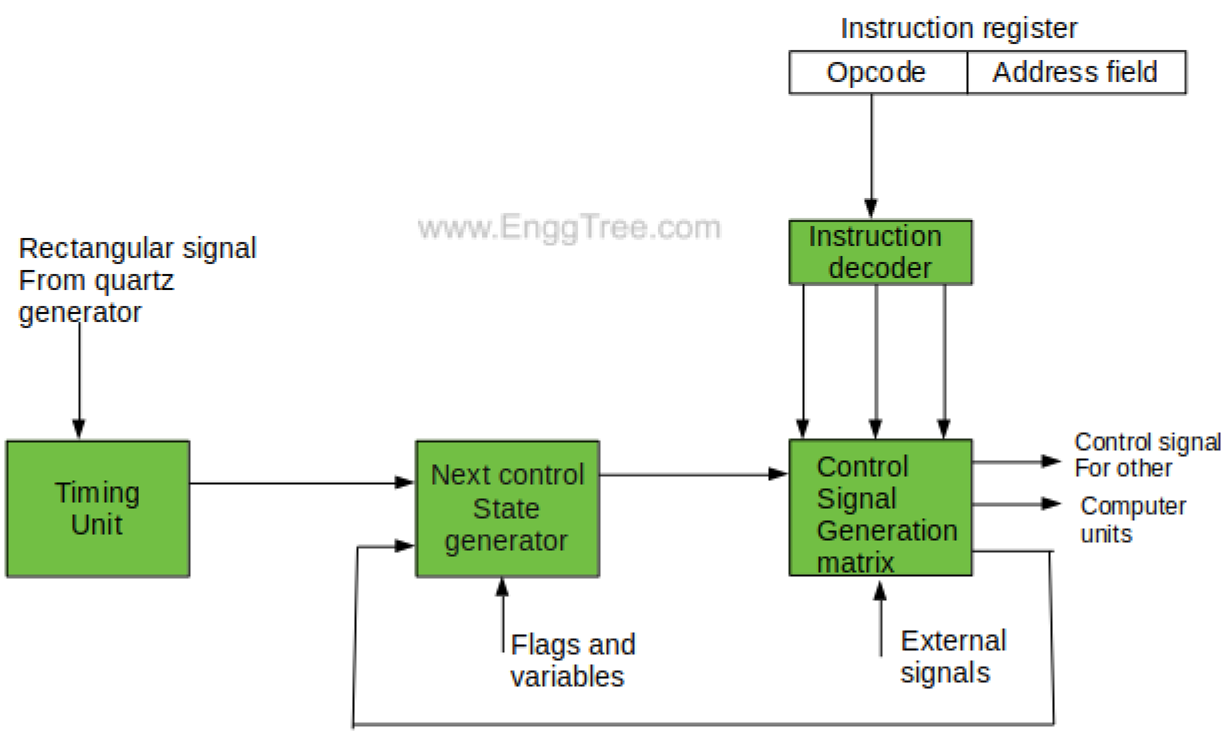
- A Hard-wired Control consists of two decoders, a sequence counter, and a number of logic gates.
- An instruction fetched from the memory unit is placed in the instruction register (IR).
- The component of an instruction register includes; I bit, the operation code, and bits 0 through 11.
- The operation code in bits 12 through 14 are coded with a 3 x 8 decoder.
- The outputs of the decoder are designated by the symbols D0 through D7.
- The operation code at bit 15 is transferred to a flip-flop designated by the symbol I.
- The operation codes from Bits 0 through 11 are applied to the control logic gates.
- The Sequence counter (SC) can count in binary from 0 through 15



Control Unit of a Basic Computer

In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we can not modify the signal generation method without physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode.

As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for execution units of the computer. This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays.


Instruction register
Opcode
Address field
www.EnggTree.com
Instruction decoder
Rectangular signal From quartz generator
Timing Unit
Next control State generator
Control Signal Generation matrix
Control signal For other Computer units
Flags and variables
External signals

Block diagram of a hardwired control unit of a computer

Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle. Following the structure of this cycle, the suitable sequence of internal states is organized in the control unit.

A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix. This matrix combines these signals with the timing signals, which are generated by the timing unit based on the rectangular patterns usually supplied by the quartz generator. When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching. Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered. A change of any of the earlier mentioned signals stimulates the change of the control unit state.

This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing). The values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle.

The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of computer. When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.

Pros(advantage) of Hardwired Control Unit

- Hardwired Control Unit is quick due to the usage of combinational circuits to generate signals.
- The amount of delay that can occur in the creation of control signals is dependent on the number of gates.
- It can be tweaked to get the fastest mode of operation.
- Quicker than a micro-programmed control unit.

Cons(disadvantage) of Hardwired Control Unit

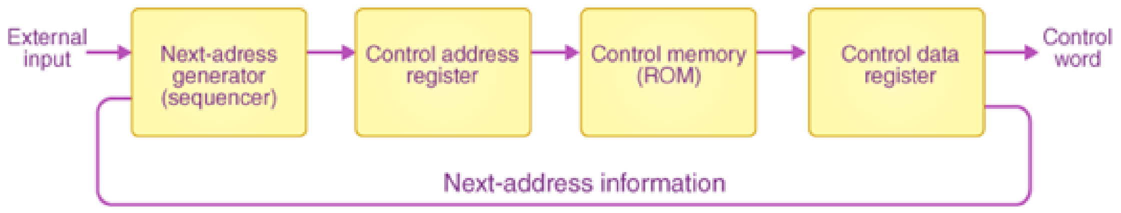
- As it require additional control signals to be created, the design becomes morecomplex (need for more encoders or decoders).
- Changes to control signals are challenging since they necessitate rearrangingwires in the hardware circuit.
- It’s difficult and time-consuming to add a new feature.
- It’s difficult to evaluate and fix flaws in the initial design.

MICRO PROGRAMMED CONTROL METHOD

It is programming method to design control unit.

It consists of micro program stored in control memory called ROM.

Micro program consists of large number of micro instructions. When executing a micro instruction it will generate a appropriate control signal.



Microprogrammed control unit of a Basic Computer

Next address generator examine the instruction within Instruction Register(IR) and find the address of micro instruction within micro program to create the corresponding control signal.

Send the micro instruction address to control address register. Now control address register contain the address of microinstruction.

Then the appropriate microinstruction is pick from the control memory (ROM). (Control memory contains the micro program)

The micro instruction is then shifted to control data register.

The data processor executes the microinstruction to produce the control signal.

While executing the micro instruction, next address generator generate the address of the micro instruction needed to execute next.

The address of micro instruction stored control address register(micro instruction address register) is decoded then it given as input to control memory(control store)

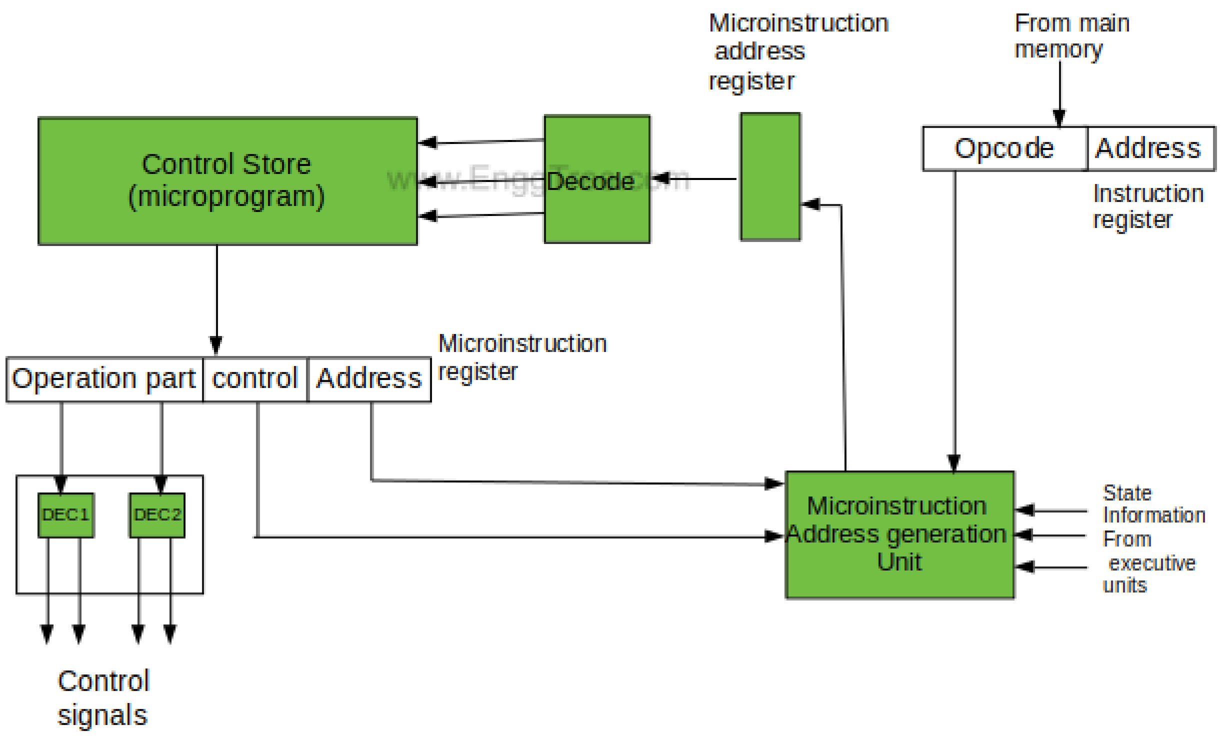
Micro instructions operation part is decoded and produce the control signals.

Micro instructions control part and address part is useful to generate the next micro instruction address and provide this information to next address generator.

The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution.

In microprogrammed control units, subsequent instruction words are fetched into the instruction register in a normal way. However, the operation code of each instruction is not directly decoded to enable immediate control signal generation but it comprises the initial address of a microprogram contained in the control store.

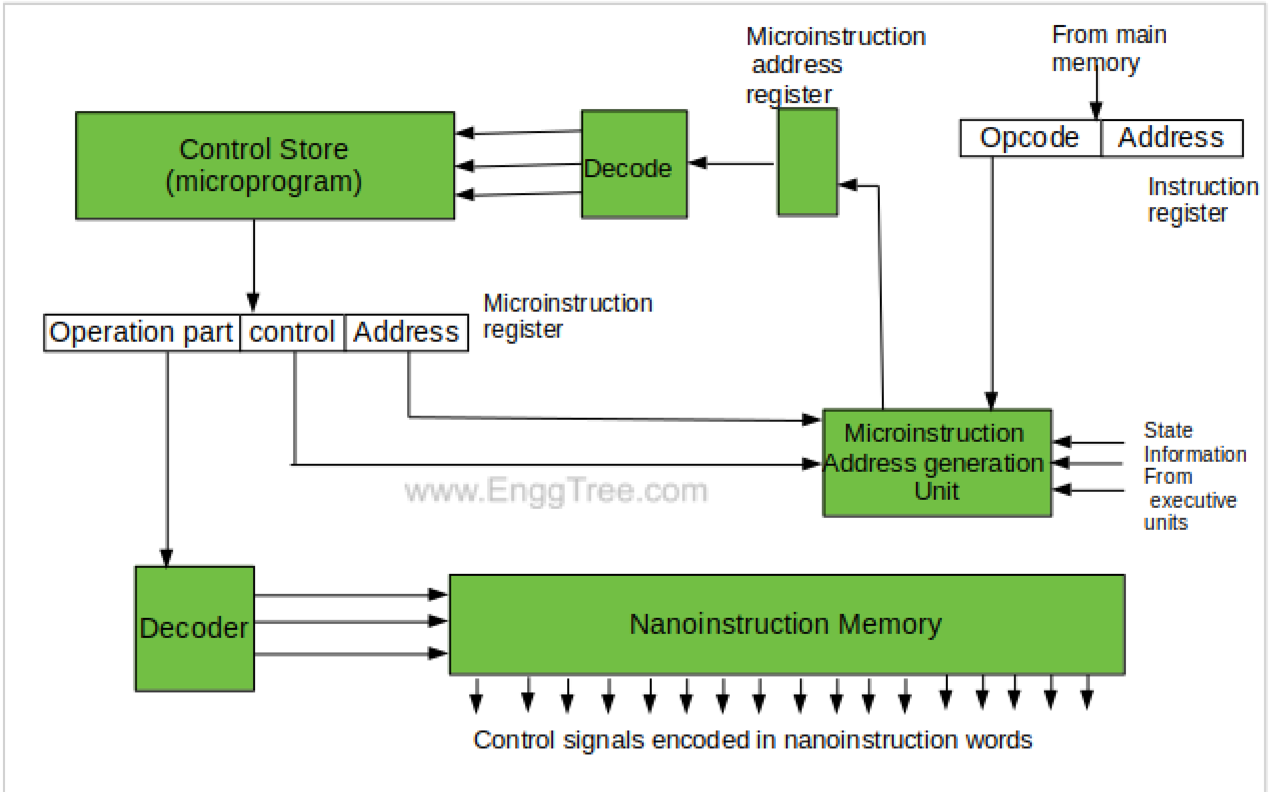
- **With a single-level control store:**
  In this, the instruction opcode from the instruction register is sent to the control store address register. Based on this address, the first microinstruction of a microprogram that interprets execution of this instruction is read to the microinstruction register. This microinstruction contains in its operation part encoded control signals, normally as few bit fields. In a set microinstruction field decoders, the fields are decoded. The microinstruction also contains the address of the next microinstruction of the given instruction microprogram and a control field used to control activities of the microinstruction address generator.


Microinstruction address register
From main memory
Control Store (microprogram)
Decode
Opcode
Address
Instruction register
Operation part
control
Address
Microinstruction register
DEC1
DEC2
Microinstruction Address generation Unit
State Information From executive units
Control signals

Microprogrammed control unit with a single level control store

The last mentioned field decides the addressing mode (addressing operation) to be applied to the address embedded in the ongoing microinstruction. In microinstructions along with conditional addressing mode, this address is refined by using the processor condition flags that represent the status of computations in the current program. The last microinstruction in the instruction of the given microprogram is the microinstruction that fetches the next instruction from the main memory to the instruction register.

- **With a two-level control store:**
  In this, in a control unit with a two-level control store, besides the control memory for microinstructions, a nano-instruction memory is included. In such a control unit, microinstructions do not contain encoded control signals. The operation part of microinstructions contains the address of the word in the nano-instruction memory, which contains encoded control signals. The nano-instruction memory contains all combinations of control signals that appear in microprograms that interpret the complete instruction set of a given computer, written once in the form of nano-instructions.


Microinstruction address register
From main memory
Control Store (microprogram)
Decode
Opcode
Address
Instruction register
Operation part
control
Address
Microinstruction register
Microinstruction Address generation Unit
State Information From executive units
www.EnggTree.com
Decoder
Nanoinstruction Memory
Control signals encoded in nanoinstruction words

Microprogrammed control unit with a two-level control store

In this way, unnecessary storing of the same operation parts of microinstructions is avoided. In this case, microinstruction word can be much shorter than with the single level control store. It gives a much smaller size in bits of the microinstruction memory and, as a result, a much smaller size of the entire control memory. The microinstruction memory contains the control for selection of consecutive microinstructions, while those control signals are generated at the basis of nano-instructions. In nano-instructions, control signals are frequently encoded using 1 bit/ 1 signal method that eliminates decoding.

Micro programmed Control Unit Pros(advantage)

- It allows for a more methodical control unit design.
- It’s easier to troubleshoot and modify.

- It can keep the control function's fundamental structure.
- It can make the control unit's design easier. As a result, it is less expensive and less prone to errors or glitches.
- It has the ability to design in a methodical and ordered manner.
- It is used to control software-based functions rather than hardware-based functions.
- It's more adaptable.
- It is used to do complex functions with ease.

Microprogrammed Control Unit Cons(disadvantage)

- Adaptability comes at a higher price.
- It is comparatively slower than a control unit that is hardwired.

Differences between hardwired control and micro programmed control

Hardwired control	**Micro programmed control**
i. It is a hardware based method to design control unit	It is programming method to design control unit
ii. Here control logic circuits generate the control signals	Here micro instructions generate the control signals
iii. It is difficult one to modify the method of generating control signals	It is easier one to modify
iv. Higher speed than micro programmed control method	It is slower than hardwired control
v. Costlier than micro program method	cheaper than hardwired method

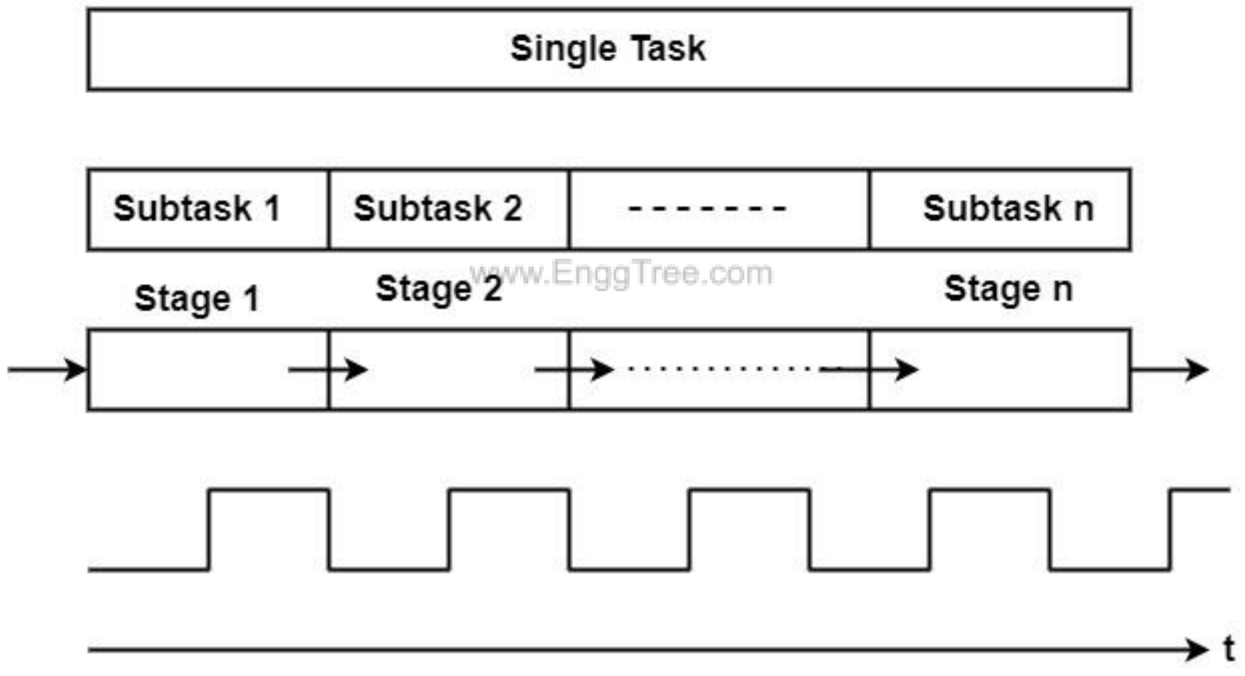
PIPELINING

Pipelining defines the temporal overlapping of processing. Pipelines are emptiness greater than assembly lines in computing that can be used either for instruction processing or, in a more general method, for executing any complex operations. It can be used efficiently only for a sequence of the same task, much similar to assembly lines.

A basic pipeline processes a sequence of tasks, including instructions, as per the following principle of operation −

Each task is subdivided into multiple successive subtasks as shown in the figure. For instance, the execution of register-register instructions can be broken down into instruction fetch, decode, execute, and writeback.

Basic Principle of Pipelining



A pipeline phase related to each subtask executes the needed operations.

A similar amount of time is accessible in each stage for implementing the needed subtask.

All pipeline stages work just as an assembly line that is, receiving their input generally from the previous stage and transferring their output to the next stage.

Finally, it can consider the basic pipeline operates clocked, in other words synchronously. This defines that each stage gets a new input at the beginning of the clock cycle, each stage has a single clock cycle available for implementing the needed operations, and each stage produces the result to the next stage by the starting of the subsequent clock cycle.

Five phases of instruction execution

1. Instruction Fetch
2. Instruction decode
3. Instruction execution
4. Memory access
5. Write back

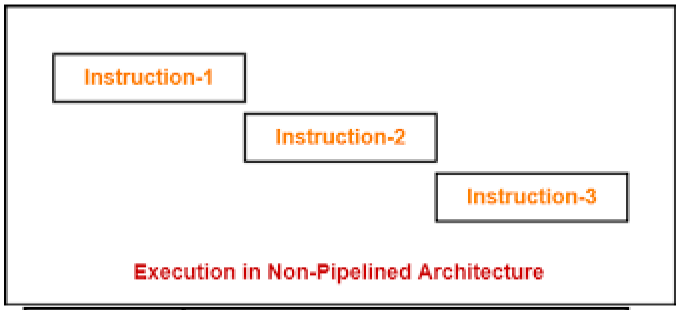
Pipelining take this instruction execution phases into pipelining stages.

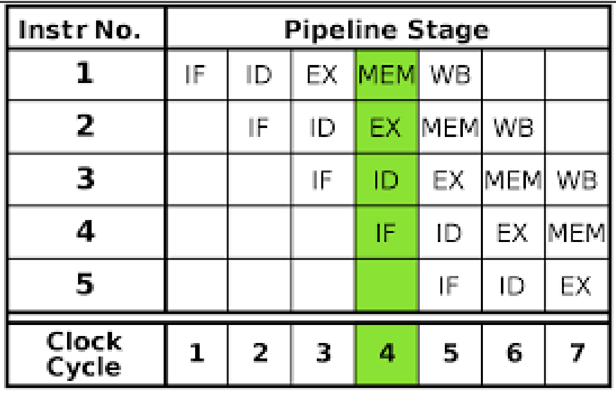
Pipelining stages

i. Instruction Fetch(IF)
ii. Instruction Decode (ID)
iii. Instruction Execution (ALU)
iv. Memory Access (MA)
v. Write Back(WB)

Pipelining is used to improve the program execution speed. It never leave the processor components in idle position.

Each stage in pipeline is executed in one clock cycle of CPU.





Instr No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

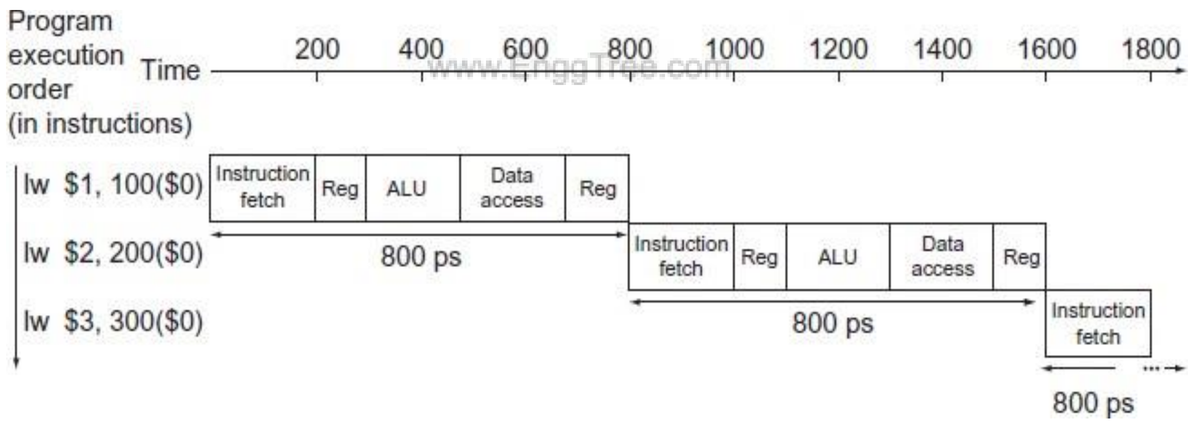
Pipelined Execution

$$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instruction}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

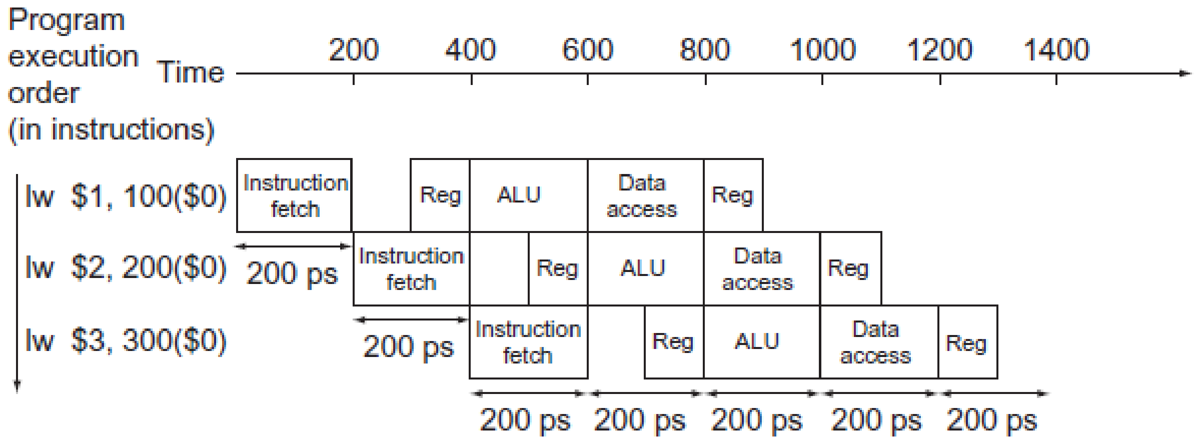
Exact time required to finish each stage of pipeline in MIPS processor is given as follows

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, AND, OR, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

Following diagram shows time between instruction in non pipelined execution and pipelined execution.



Non- Pipelined Execution



Pipelined execution

Types of pipelines

There are two types of pipelines in computer processing.

Instruction pipeline

The instruction pipeline represents the stages in which an instruction is moved through the various segments of the processor, starting from fetching and then buffering, decoding and executing. One segment reads instructions from the memory, while, simultaneously, previous instructions are executed in other segments. Since these processes happen in an overlapping manner, the throughput of the entire system increases. The pipeline's efficiency can be further increased by dividing the instruction cycle into equal-duration segments.

Pipeline processing can occur not only in the data stream but in the instruction stream as well.

Most of the digital computers with complex instructions require instruction pipeline to carry out operations like fetch, decode and execute instructions.

In general, the computer needs to process each instruction with the following sequence of steps.

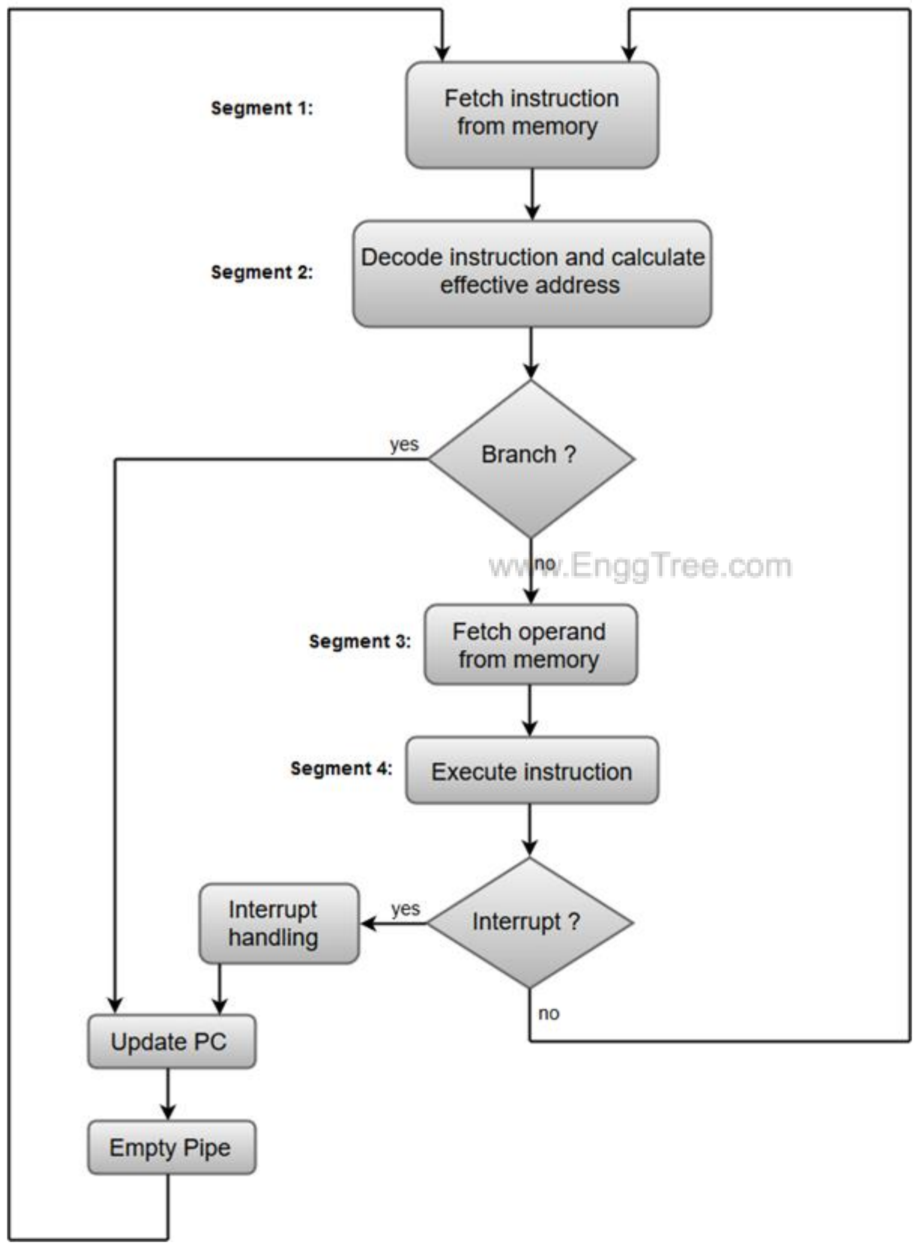
1. Fetch instruction from memory.
2. Decode the instruction.
3. Calculate the effective address.
4. Fetch the operands from memory.
5. Execute the instruction.
6. Store the result in the proper place.

Each step is executed in a particular segment, and there are times when different segments may take different times to operate on the incoming information. Moreover, there are times when two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.

The organization of an instruction pipeline will be more efficient if the instruction cycle is divided into segments of equal duration. One of the most common examples of this type of organization is a **Four-segment instruction pipeline.**

A **four-segment instruction** pipeline combines two or more different segments and makes it as a single one. For instance, the decoding of the instruction can be combined with the calculation of the effective address into one segment.

The following block diagram shows a typical example of a four-segment instruction pipeline. The instruction cycle is completed in four segments.



Segment 1:

The instruction fetch segment can be implemented using first in, first out (FIFO) buffer.

Segment 2:

The instruction fetched from memory is decoded in the second segment, and eventually, the effective address is calculated in a separate arithmetic circuit.

Segment 3:

An operand from memory is fetched in the third segment.

Segment 4:

The instructions are finally executed in the last segment of the pipeline organization.

Arithmetic pipeline

Arithmetic Pipelines are mostly used in high-speed computers. They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems.

To understand the concepts of arithmetic pipeline in a more convenient way, let us consider an example of a pipeline unit for floating-point addition and subtraction.

The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers defined as:

$X = A * 2^a = 0.9504 * 10^3$
$Y = B * 2^b = 0.8200 * 10^2$

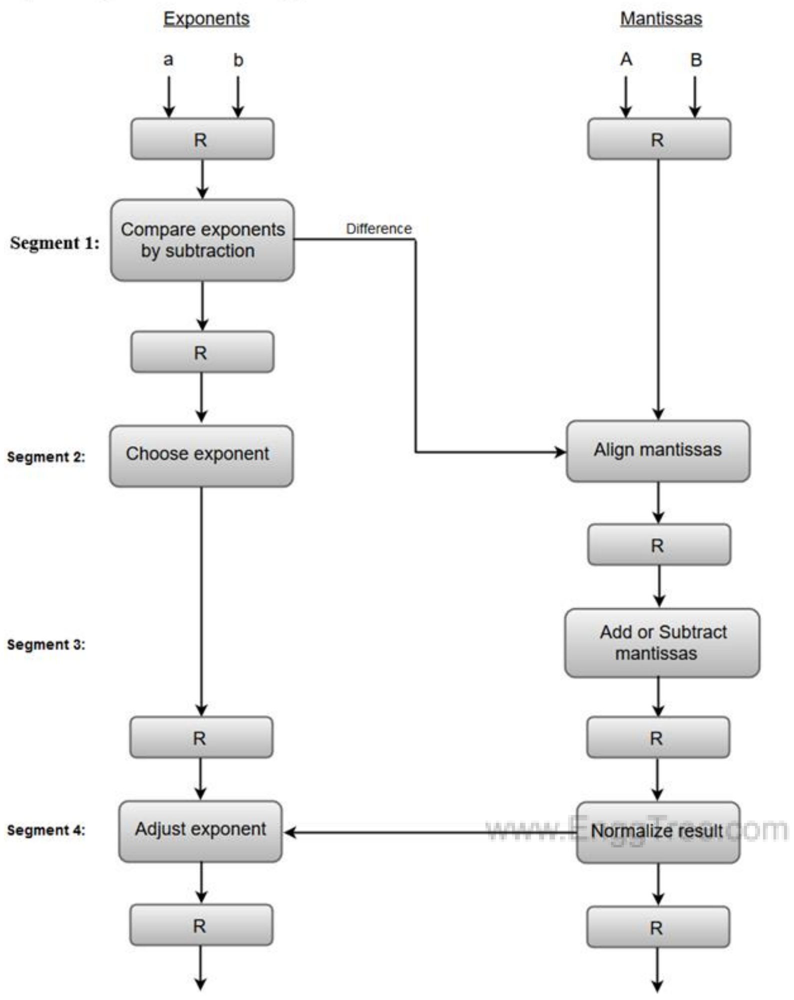
Where **A** and **B** are two fractions that represent the mantissa and **a** and **b** are the exponents.

The combined operation of floating-point addition and subtraction is divided into four segments. Each segment contains the corresponding suboperation to be performed in the given pipeline. The suboperations that are shown in the four segments are:

1. Compare the exponents by subtraction.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

The following block diagram represents the suboperations performed in each segment of the pipeline.

Pipeline organization for floating point addition and subtraction:



Note: Registers are placed after each suboperation to store the intermediate results.

1. Compare exponents by subtraction:

The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.

The difference of the exponents, i.e., **3 - 2** = **1** determines how many times the mantissa associated with the smaller exponent must be shifted to the right.

2. Align the mantissas:

The mantissa associated with the smaller exponent is shifted according to the difference of exponents determined in segment one.

$X = 0.9504 * 10^3$
$Y = 0.08200 * 10^3$

3. Add mantissas:

The two mantissas are added in segment three.

$Z = X + Y = 1.0324 * 10^3$

4. Normalize the result:

After normalization, the result is written as:

$Z = 0.1324 * 10^4$

Advantages of Pipelining

- The cycle time of the processor is decreased. It can improve the instruction throughput. Pipelining doesn't lower the time it takes to do an instruction. Rather than, it can raise the multiple instructions that can be processed together ("at once") and lower the delay between completed instructions (known as 'throughput').
- If pipelining is used, the CPU Arithmetic logic unit can be designed quicker, but more complex.
- Pipelining increases execution over an un-pipelined core by an element of the multiple stages (considering the clock frequency also increases by a similar factor) and the code is optimal for pipeline execution.
- Pipelined CPUs frequently work at a higher clock frequency than the RAM clock frequency, (as of 2008 technologies, RAMs operate at a low frequency correlated to CPUs frequencies) increasing the computer's global implementation.

Disadvantages of Pipelining

Designing of the pipelined processor is complex. Instruction latency increases in pipelined processors. The throughput of a pipelined processor is difficult to predict. The longer the pipeline, worse the problem of hazard for branch instructions.

PIPELINE HAZARD

There are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called *hazards*, and there are three different types.

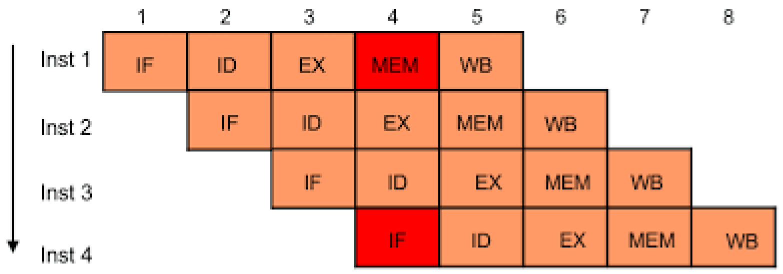
Dependencies between instructions in pipeline is called hazards
.Three types

i. Structural hazard
ii. Data hazard
iii. Control hazard

Structural hazard

Hardware dependencies between instructions in pipeline are called as structural hazard. Here hardware means memory, ALU, a register.

Example:



Here instruction1 and instruction 4 are use the memory at the same. So memory conflict will occur. At a time memory process only one work. Not able to do more than one work at a time (mutual exclusion property).

Pipeline stall

Delay in execution of instruction in pipeline is called pipeline stall. It is also called as bubble –nickname.

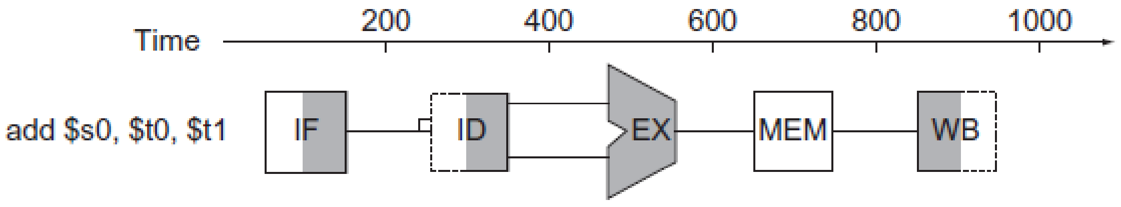
Data hazard

Data hazard occur when the pipeline must be stalled because one step must

Wait for another to complete. Also called as pipeline data hazard. When a planned instruction can not execute in proper clock cycle because data that is need to execute the instruction is not yet available

Consider following two instructions

```
add   $s0, $t0, $t1
sub   $t2, $s0, $t3
```

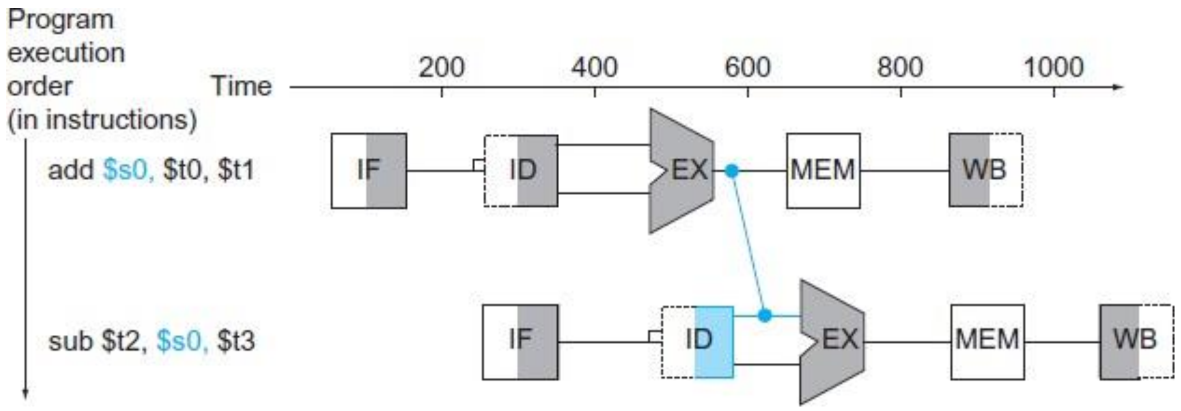


Subtract operation needs the value of S0. But S0 value available write at WB stage.

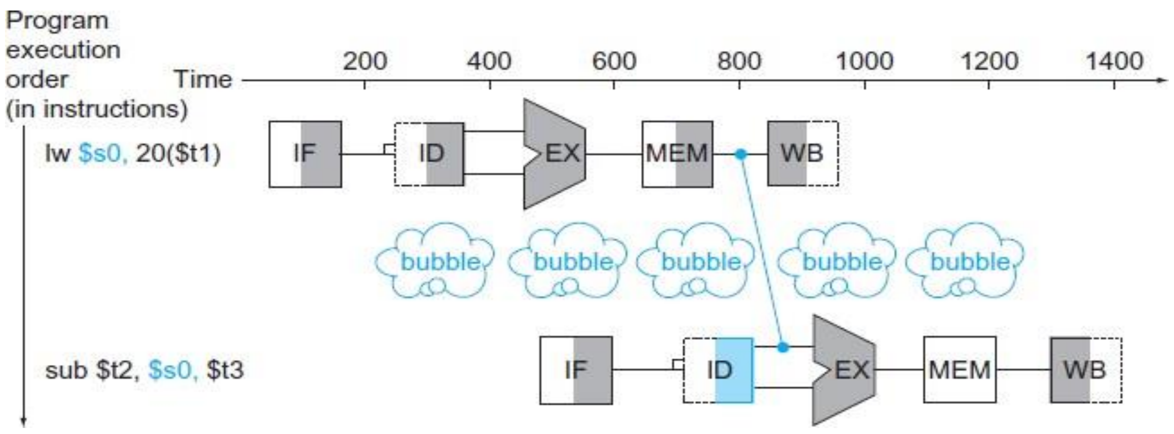
Forwarding

The primary solution is based on the observation that we don't need to wait for the instruction to complete before trying to resolve the data hazard. For the code sequence above, as soon as the ALU creates the sum for the add, we can supply it as an input for the subtract. Adding extra hardware to retrieve the missing item early from the internal resources is called forwarding or bypassing.

Forwarding technique is useful to forward the results from internal hardware units.

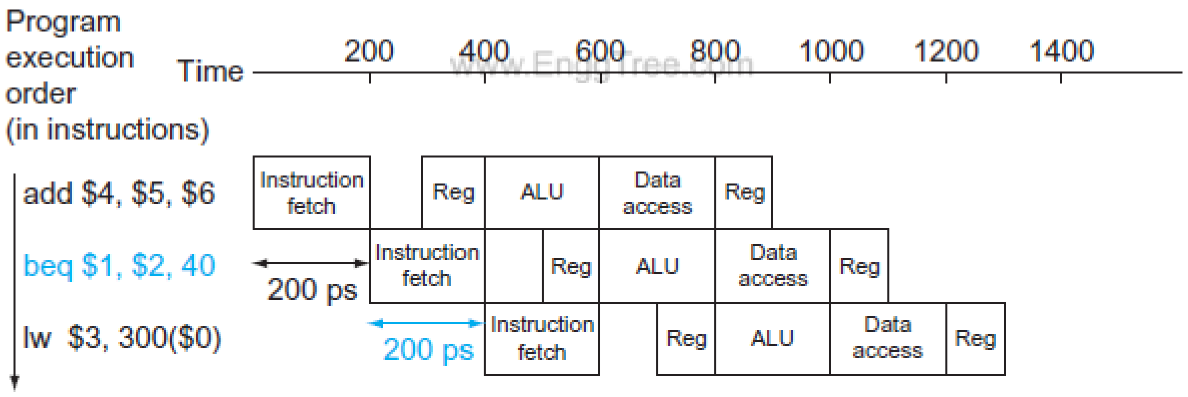


Forward another example:



Control hazard

Control hazard arising from the need to make a decision based on the results of one instruction while others are executing.



In the above example , if $1 = $2 then go to the branch address PC=PC + 40x4

But here next load word instruction, lw $3, 300($0) is fetched from memory.

Suppose branch is taken then the pipeline wrongly fetch and execute the load word instruction

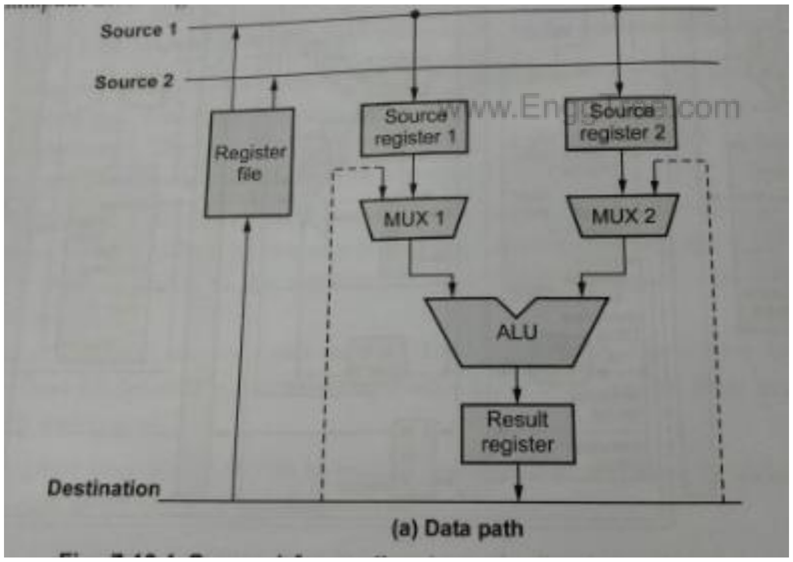
HANDLING DATA HAZARDS & CONTROL HAZARDS

Hazards: Prevent the next instruction in the instruction stream from executing during its designated clock cycle.
* Hazards reduce the performance from the ideal speedup gained by pipelining.
3 classes of hazards:

Ø Structural hazards: arise from resource conflicts when the hardware cannot support all possible combinations of instructions simultaneously in overlapped execution.

Ø Data hazards: arise when an instruction depends on the results of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.

Ø Control hazards: arise from the pipelining of branches and other instructions that change the PC.



(a) Data path

Performance of Pipelines with Stalls

* A stall causes the pipeline performance to degrade from the ideal performance.

Speedup from pipelining = [1/ (1+ pipeline stall cycles per instruction)] * Pipeline

$$\text{Speedup from pipelining} = \frac{1}{1+\text{ pipeline stall cycles per instruction}} * \text{Pipeline depth}$$

Structural Hazards

* When a processor is pipelined, the overlapped execution of instructions requires pipelining of functional units and duplication of resources to allow all possible combinations of instructions in the pipeline.

* If some combination of instructions cannot be accommodated because of resource conflicts, the processor is said to have a structural hazard.
* Instances:

  § When functional unit is not fully pipelined, Then a sequence of instructions using that unpipelined unit cannot proceed at the rate of one per clock cycle.

  § when some resource has not been duplicated enough to allow all combinations of instructions in the pipeline to execute.

* To Resolve this hazard,

  § Stall the the pipeline for 1 clock cycle when the data memory access occurs. A stall is commonly called a pipeline bubble or just bubble, since it floats through the pipeline taking space but carrying no useful work.

Data Hazards

* A major effect of pipelining is to change the relative timing of instructions by overlapping their execution. This overlap introduces data and control hazards.

* Data hazards occur when the pipeline changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing instructions on an unpipelined processor.

Minimizing Data Hazard Stalls by Forwarding

* The problem solved with a simple hardware technique called forwarding (also called bypassing and sometimes short-circuiting). Forwards works as:

  § The ALU result from both the EX/MEM and MEM/WB pipeline registers is always fed back to the ALU inputs.

  § If the forwarding hardware detects that the previous ALU operation has written the register corresponding to a source for the current ALU operation, control logic selects the forwarded result as the ALU input rather than the value read from the register file.

Data Hazards Requiring Stalls

* The load instruction has a delay or latency that cannot be eliminated by forwarding alone. Instead, we need to add hardware, called a pipeline interlock, to preserve the correct execution pattern.

* A pipeline interlock detects a hazard and stalls the pipeline until the hazard is cleared.

* This pipeline interlock introduces a stall or bubble. The CPI for the stalled instruction increases by the length of the stall.

Branch Hazards

* Control hazards can cause a greater performance loss for our MIPS pipeline . When a branch is executed, it may or may not change the PC to something other than its current value plus 4.

* If a branch changes the PC to its target address, it is a taken branch; if it falls through, it is not taken, or untaken.

Reducing Pipeline Branch Penalties

* Simplest scheme to handle branches is to freeze or flush the pipeline, holding or deleting any instructions after the branch until the branch destination is known.

* A higher-performance, and only slightly more complex, scheme is to treat every branch as not taken, simply allowing the hardware to continue as if the branch were not

executed. The complexity of this scheme arises from having to know when the state might be changed by an instruction and how to “back out” such a change.

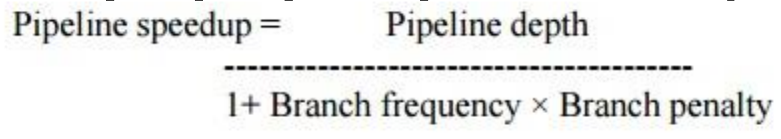
* In simple five-stage pipeline, this predicted-not-taken or predicted untaken scheme is implemented by continuing to fetch instructions as if the branch were a normal instruction.

§ The pipeline looks as if nothing out of the ordinary is happening.

§ If the branch is taken, however, we need to turn the fetched instruction into a no-op and restart the fetch at the target address.

* An alternative scheme is to treat every branch as taken. As soon as the branch is decoded and the target address is computed, we assume the branch to be taken and begin fetching and executing at the target**Performance of Branch Schemes**

Pipeline speedup = Pipeline depth / [1+ Branch frequency × Branch penalty]



$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

The branch frequency and branch penalty can have a component from both unconditional and conditional branches.

UNIT V

MEMORY AND I/O

MEMORY HIERARCHY IN COMPUTER ARCHITECTURE

In the design of the computer system, **a processor**, as well as a large amount of memory devices, has been used. However, the main problem is, these parts are expensive. So the **memory organization** of the system can be done by memory hierarchy. It has several levels of memory with different performance rates. But all these can supply an exact purpose, such that the access time can be reduced. The memory hierarchy was developed depending upon the behavior of the program. This article discusses an overview of the memory hierarchy in computer architecture.

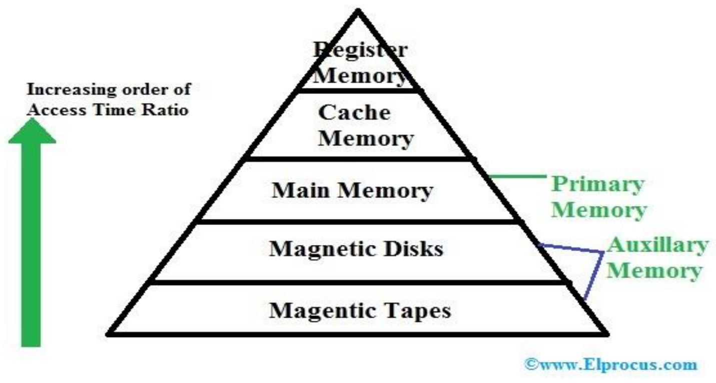
What is Memory Hierarchy?

The memory in a computer can be divided into five hierarchies based on the speed as well as use. The processor can move from one level to another based on its requirements. The five hierarchies in the memory are registers, cache, main memory, magnetic discs, and magnetic tapes. The first three hierarchies are volatile memories which mean when there is no power, and then automatically they lose their stored data. Whereas the last two hierarchies are not volatile which means they store the data permanently.

A memory element is the set of storage devices which stores the binary data in the type of bits. In general, the storage of memory can be classified into two categories such as volatile as well as non- volatile.

Memory Hierarchy in Computer Architecture

The **memory hierarchy design** in a computer system mainly includes different storage devices. Most of the computers were inbuilt with extra storage to run more powerfully beyond the main memory capacity. The following **memory hierarchy diagram** is a hierarchical pyramid for computer memory. The designing of the memory hierarchy is divided into two types such as primary (Internal) memory and secondary (External) memory.



Memory Hierarchy

Primary Memory
The primary memory is also known as internal memory, and this is accessible by the processor straightly. This memory includes main, cache, as well as CPU registers.

Secondary Memory
The secondary memory is also known as external memory, and this is accessible by the processor through an input/output module. This memory includes an optical disk, magnetic disk, and magnetic tape.

Characteristics of Memory Hierarchy

The memory hierarchy characteristics mainly include the following.

Performance
Previously, the designing of a computer system was done without memory hierarchy, and the speed gap among the main memory as well as the CPU registers enhances because of the huge disparity in access time, which will cause the lower performance of the system. So, the enhancement was mandatory. The enhancement of this was designed in the memory hierarchy model due to the system's performance increase.

Ability
The ability of the memory hierarchy is the total amount of data the memory can store. Because whenever we shift from top to bottom inside the memory hierarchy, then the capacity will increase.

Access Time
The access time in the memory hierarchy is the interval of the time among the data availability as well as request to read or write. Because whenever we shift from top to bottom inside the memory hierarchy, then the access time will increase

Cost per bit
When we shift from bottom to top inside the memory hierarchy, then the cost for each bit will increase which means an internal Memory is expensive compared with external memory.

Memory Hierarchy Design

The memory hierarchy in computers mainly includes the following.

Registers
Usually, the register is a static RAM or SRAM in the processor of the computer which is used for holding the data word which is typically 64 or 128 bits. The program counter register is the most important as well as found in all the processors. Most of the processors use a status word register as well as an accumulator. A status word register is used for decision making, and the accumulator is used to store the data like mathematical operation. Usually, computers like **complex instruction set computers** have so many registers for accepting main memory, and **RISC- reduced instruction set** computers have more registers.

Cache Memory
Cache memory can also be found in the processor, however rarely it may be another **IC (integrated circuit)** which is separated into levels. The cache holds the chunk of data which are frequently used from main memory. When the processor has a single core then it will have two (or) more cache levels rarely. Present multi-core processors will be having three, 2-levels for each one core, and one level is shared.
Main Memory
The main memory in the computer is nothing but, the memory unit in the CPU that communicates directly. It is the main storage unit of the computer. This memory is fast as well as large memory used for storing the data throughout the operations of the computer. This memory is made up of RAM as well as ROM.

Magnetic Disks
The magnetic disks in the computer are circular plates fabricated of plastic otherwise metal by magnetized material. Frequently, two faces of the disk are utilized as well as many disks may be stacked on one spindle by read or write heads obtainable on every plane. All the disks in computer turn jointly at high speed. The tracks in the computer are nothing but bits which are stored within the magnetized plane in spots next to concentric circles. These are usually separated into sections which are named as sectors.

Magnetic Tape
This tape is a normal magnetic recording which is designed with a slender magnetizable covering on an extended, plastic film of the thin strip. This is mainly used to back up huge data. Whenever the computer requires to access a strip, first it will mount to access the data. Once the data is allowed, then it will be un mounted. The access time of memory will be slower within magnetic strip as well as it will take a few minutes for accessing a strip.

Advantages of Memory Hierarchy

The need for a memory hierarchy includes the following.

- Memory distributing is simple and economical
- Removes external destruction
- Data can be spread all over
- Permits demand paging & pre-paging
- Swapping will be more proficient

Thus, this is all about **memory hierarchy**. From the above information, finally, we can conclude that it is mainly used to decrease the bit cost, access frequency, and to increase the capacity, access time.

MEMORY MANAGEMENT

What do you mean by memory management?

Memory is the important part of the computer that is used to store the data. Its management is critical to the computer system because the amount of main memory available in a computer system is very limited. At any time, many processes are competing for it. Moreover, to increase performance, several processes are executed simultaneously. For this, we must keep several processes in the main memory, so it is even more important to manage them effectively.

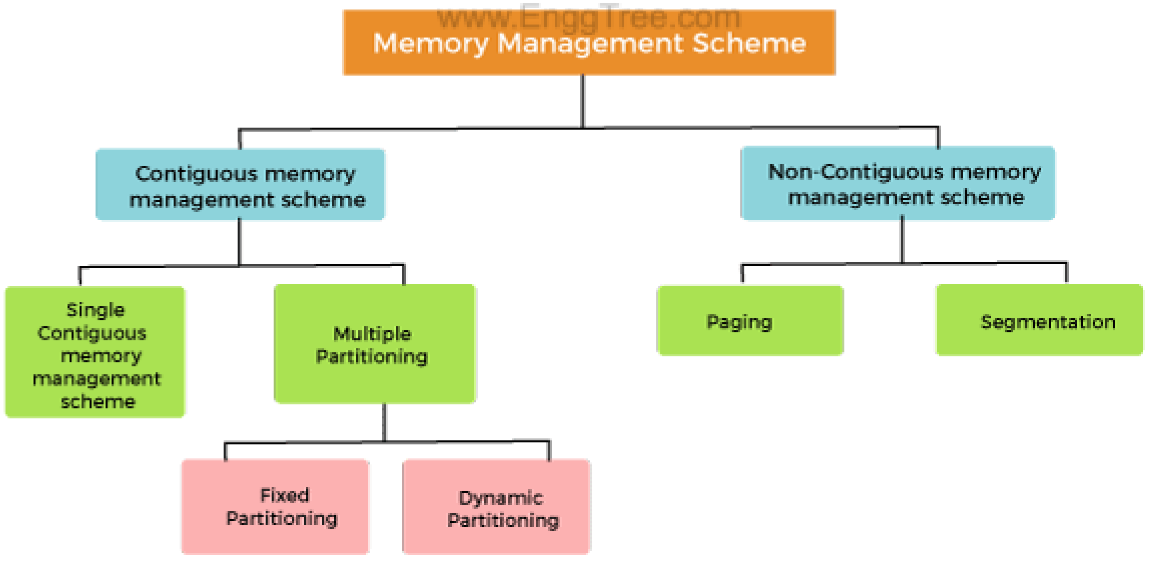
Memory management plays several roles in a computer system.

- Memory manager is used to keep track of the status of memory locations, whether it is free or allocated. It addresses primary memory by providing abstractions so that software perceives a large memory is allocated to it.
- Memory manager permits computers with a small amount of main memory to execute programs larger than the size or amount of available memory. It does this by moving information back and forth between primary memory and secondary memory by using the concept of swapping.
- The memory manager is responsible for protecting the memory allocated to each process from being corrupted by another process. If this is not ensured, then the system may exhibit unpredictable behavior.
- Memory managers should enable sharing of memory space between processes. Thus, two programs can reside at the same memory location although at different times.

Memory management Techniques:

The Memory management Techniques can be classified into following main categories:

- Contiguous memory management schemes
- Non-Contiguous memory management schemes



Classification of memory management schemes

Contiguous memory management schemes:

In a Contiguous memory management scheme, each program occupies a single contiguous block of storage locations, i.e., a set of memory locations with consecutive addresses.

Single contiguous memory management schemes:

The Single contiguous memory management scheme is the simplest memory management scheme used in the earliest generation of computer systems. In this scheme, the main memory is divided into two contiguous areas or partitions. The operating systems reside permanently in one partition, generally at the lower memory, and the user process is loaded into the other partition.

Advantages of Single contiguous memory management schemes:

- Simple to implement.
- Easy to manage and design.
- In a Single contiguous memory management scheme, once a process is loaded, it is given full processor's time, and no other processor will interrupt it.

Disadvantages of Single contiguous memory management schemes:

- Wastage of memory space due to unused memory as the process is unlikely to use all the available memory space.
- The CPU remains idle, waiting for the disk to load the binary image into the main memory.
- It can not be executed if the program is too large to fit the entire available main memory space.
- It does not support multiprogramming, i.e., it cannot handle multiple programs simultaneously.

Multiple Partitioning:

The single Contiguous memory management scheme is inefficient as it limits computers to execute only one program at a time resulting in wastage in memory space and CPU time. The problem of inefficient CPU use can be overcome using multiprogramming that allows more than one program to run concurrently. To switch between two processes, the operating systems need to load both processes into the main memory. The operating system needs to divide the available main memory into multiple parts to load multiple processes into the main memory. Thus multiple processes can reside in the main memory simultaneously.

The multiple partitioning schemes can be of two types:

- Fixed Partitioning
- Dynamic Partitioning

Fixed Partitioning

The main memory is divided into several fixed-sized partitions in a fixed partition memory management scheme or static partitioning. These partitions can be of the same size or different sizes. Each partition can hold a single process. The number of partitions determines the degree of multiprogramming, i.e., the maximum number of processes in memory. These partitions are made at the time of system generation and remain fixed after that.

Advantages of Fixed Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

Disadvantages of Fixed Partitioning memory management schemes:

- This scheme suffers from internal fragmentation.
- The number of partitions is specified at the time of system generation.

Dynamic Partitioning

The dynamic partitioning was designed to overcome the problems of a fixed partitioning scheme. In a dynamic partitioning scheme, each process occupies only as much memory as they require when loaded for processing. Requested processes are allocated memory until the entire physical memory is exhausted or the remaining space is insufficient to hold the requesting process. In this scheme the partitions used are of variable size, and the number of partitions is not defined at the system generation time.

Advantages of Dynamic Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

Disadvantages of Dynamic Partitioning memory management schemes:

- This scheme also suffers from internal fragmentation.
- The number of partitions is specified at the time of system segmentation.

Non-Contiguous memory management schemes:

In a Non-Contiguous memory management scheme, the program is divided into different blocks and loaded at different portions of the memory that need not necessarily be adjacent to one another. This scheme can be classified depending upon the size of blocks and whether the blocks reside in the main memory or not.

What is paging?

Paging is a technique that eliminates the requirements of contiguous allocation of main memory. In this, the main memory is divided into fixed-size blocks of physical memory called frames. The size of a frame should be kept the same as that of a page to maximize the main memory and avoid external fragmentation.

Advantages of paging:

- Pages reduce external fragmentation.
- Simple to implement.

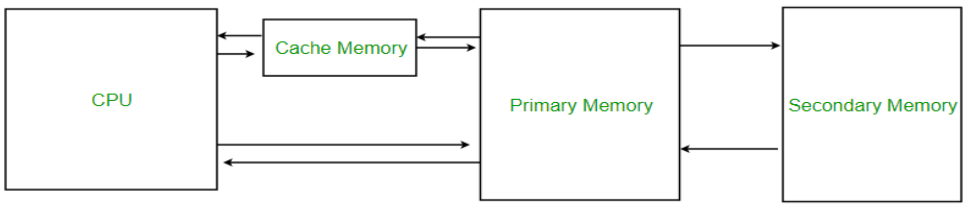
- Memory efficient.
- Due to the equal size of frames, swapping becomes very easy.
- It is used for faster access of data.

What is Segmentation?

Segmentation is a technique that eliminates the requirements of contiguous allocation of main memory. In this, the main memory is divided into variable-size blocks of physical memory called segments. It is based on the way the programmer follows to structure their programs. With segmented memory allocation, each job is divided into several segments of different sizes, one for each module. Functions, subroutines, stack, array, etc., are examples of such modules.

CACHE MEMORY

Cache Memory is a special very high-speed memory. It is used to speed up and synchronize with high-speed CPU. Cache memory is costlier than main memory or disk memory but more economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed. Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory that stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.



Levels of memory:

- **Level 1 or Register –** It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory –** It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory –** It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory –** It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance: When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from the cache.

- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**
Hit ratio = hit / (hit + miss) = no. of hits/total accesses

We can improve Cache performance using higher cache block size, and higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Cache Mapping: There are three different types of mapping used for the purpose of cache memory which is as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained below.

A. Direct Mapping

The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or In Direct mapping, assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

i = j modulo m

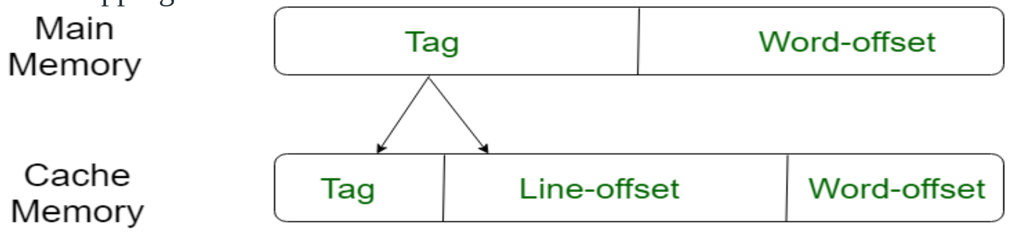
where

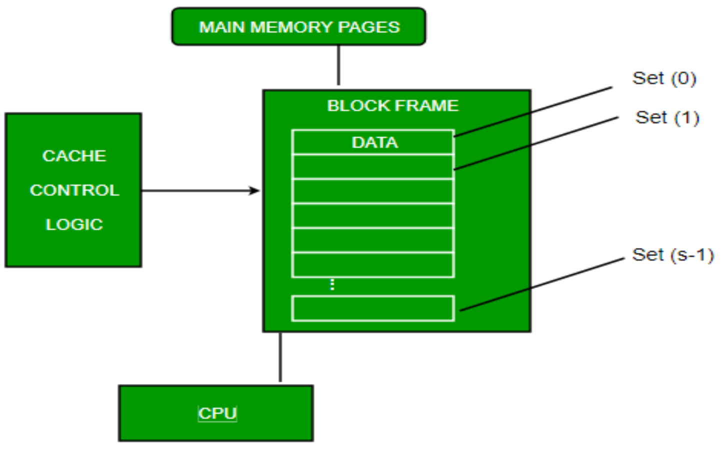
i=cache line number

j= main memory block number

m=number of lines in the cache

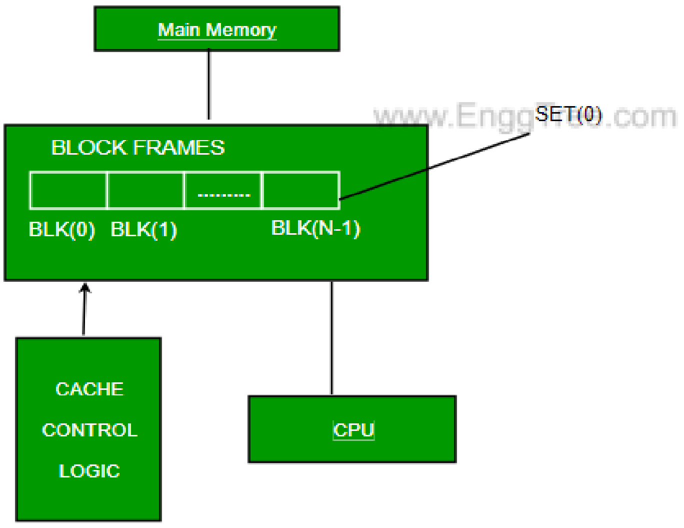
1. For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks of main memory. The cache logic interprets these s bits as a tag of s-r bits (most significant portion) and a line field of r bits. This latter field identifies one of the $m=2^r$ lines of the cache. Line offset is index bits in the direct mapping.





B. Associative Mapping

In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form. In associative mapping the index bits are zero.



C. Set-associative Mapping

This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***. Then a block in memory can map to any one of the lines of a specific set. Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques. In set associative mapping the index bits are given by the set offset bits. In this case, the cache consists of a number of sets, each of which consists of a number of lines. The relationships are m = v * k

i= j mod v

where

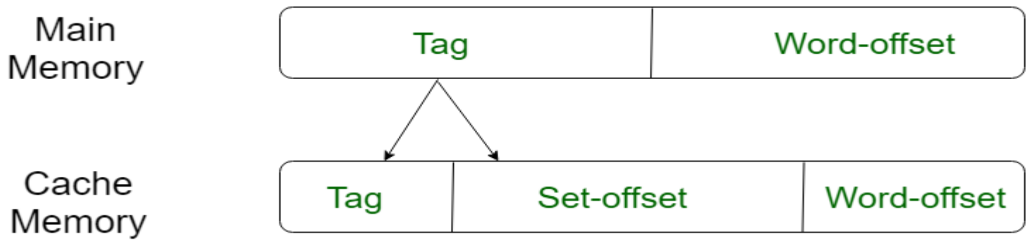
i=cache set number

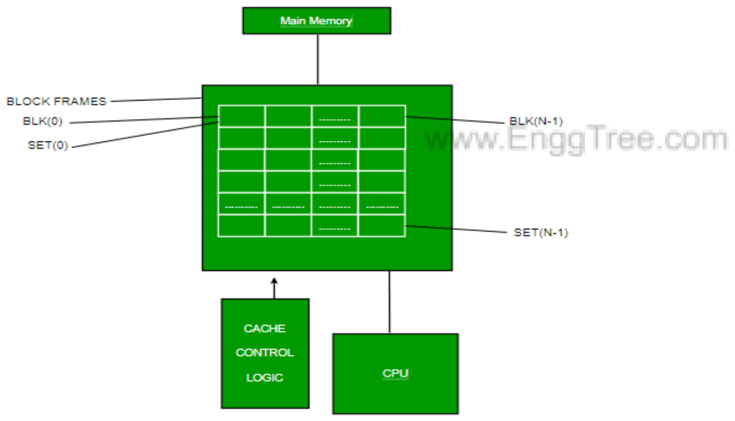
j=main memory block number

v=number of sets

m=number of lines in the cache number of sets

k=number of lines in each set





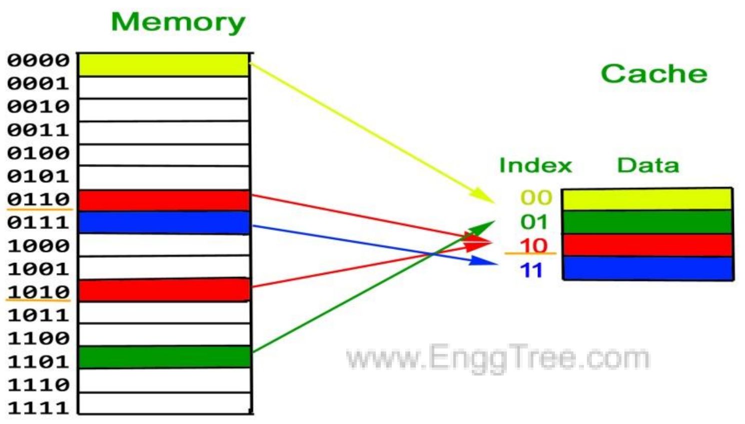
Application of Cache Memory:

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.
3. **Primary Cache** – A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
4. **Secondary Cache** – Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.
5. **Spatial Locality of reference** This says that there is a chance that the element will be present in close proximity to the reference point and next time if again searched then more close proximity to the point of reference.
6. **Temporal Locality of reference** In this Least recently used algorithm will be used. Whenever there is page fault occurs within a word will not only load word in main memory but complete page fault will be loaded because the spatial locality of reference

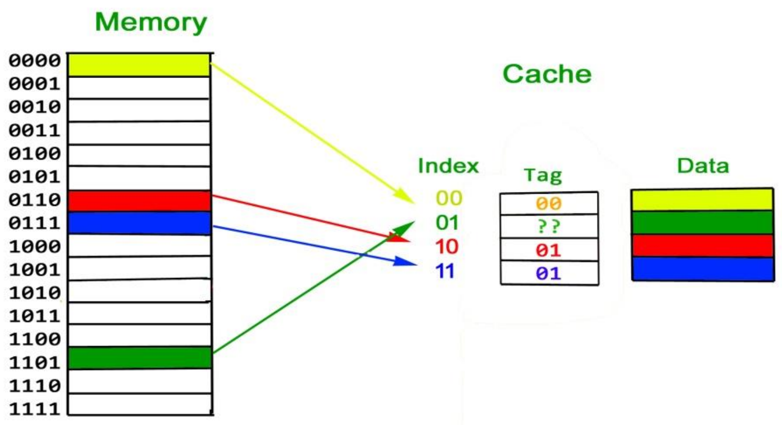
rule says that if you are referring to any word next word will be referred to in its register that's why we load complete page table so the complete block will be loaded.

Cache Organization

Cache is close to CPU and faster than main memory. But at the same time is smaller than main memory. The cache organization is about mapping data in memory to a location in cache. **A Simple Solution:** One way to go about this mapping is to consider last few bits of long memory address to find small cache address, and place them at the found address. **Problems With Simple Solution:** The problem with this approach is, we lose the information about high order bits and have no way to find out the lower order bits belong to which higher order bits.

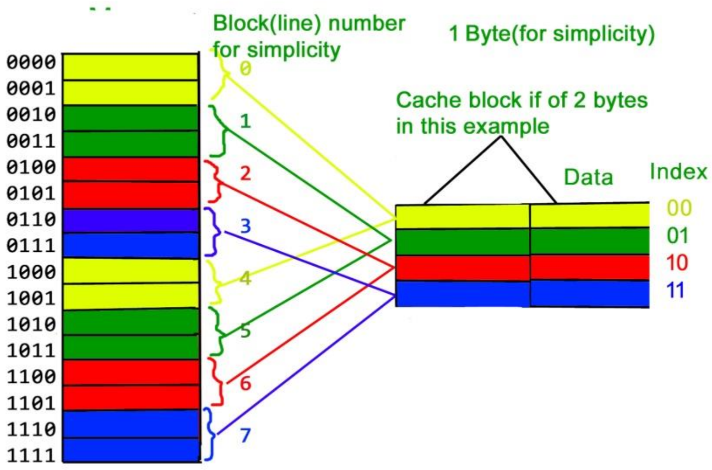


Solution is Tag: To handle above problem, more information is stored in cache to tell which block of memory is stored in cache. We store additional information as **Tag**

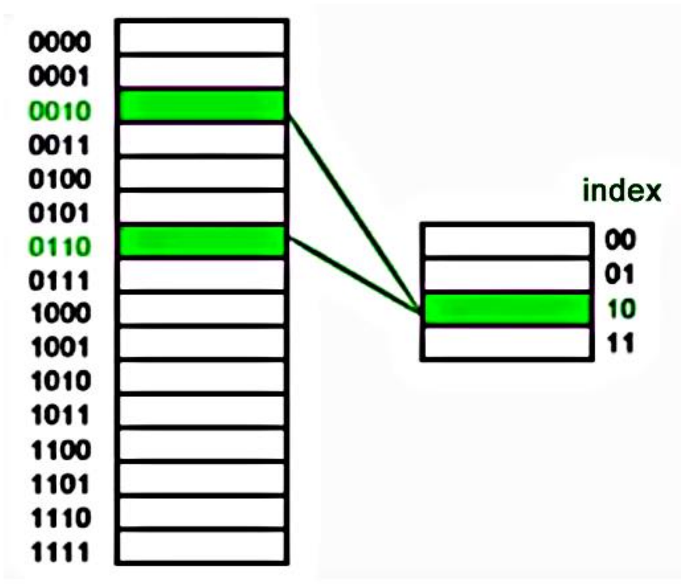


What is a Cache Block? Since programs have Spatial Locality (Once a location is retrieved, it is highly probable that the nearby locations would be retrieved in near future).

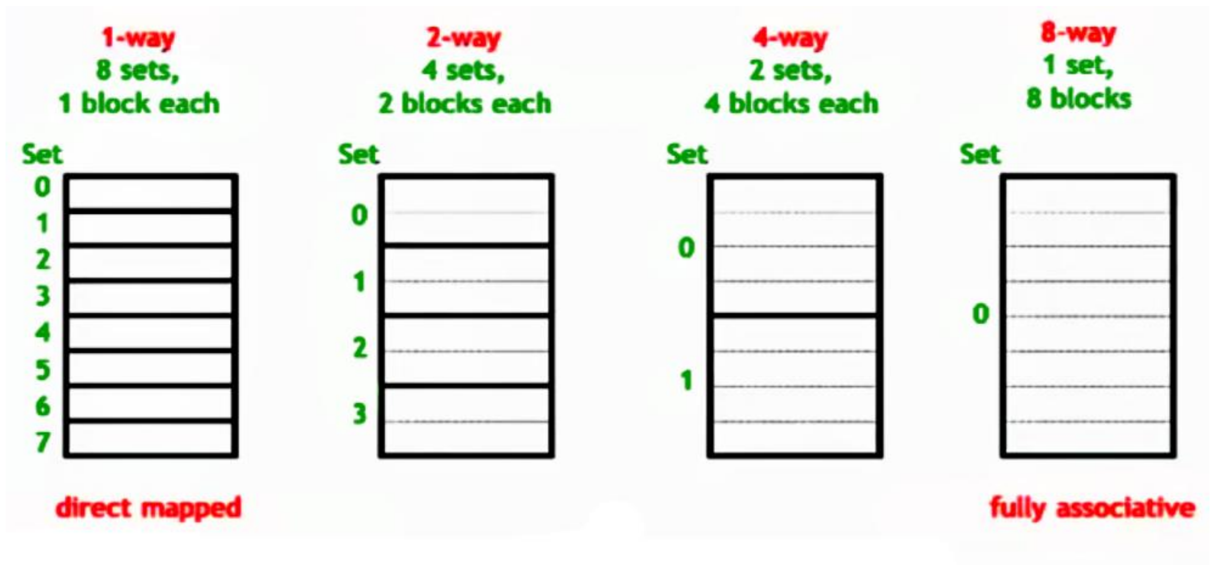
So a cache is organized in the form of blocks. Typical cache block sizes are 32 bytes or 64 bytes.



The above arrangement is Direct Mapped Cache and it has following problem We have discussed above that last few bits of memory addresses are being used to address in cache and remaining bits are stored as tag. Now imagine that cache is very small and addresses of 2 bits. Suppose we use the last two bits of main memory address to decide the cache (as shown in below diagram). So if a program accesses 2, 6, 2, 6, 2, …, every access would cause a hit as 2 and 6 have to be stored in same location in cache.



Solution to above problem – Associativity What if we could store data at any place in cache, the above problem won't be there? That would slow down cache, so we do something in between.



VIRTUAL MEMORY

Virtual Memory

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.

Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.

By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.

How Virtual Memory Works?

In modern word, virtual memory has become quite common these days. In this scheme, whenever some pages needs to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory.

Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.

Demand Paging

Demand Paging is a popular method of virtual memory management. In demand paging, the pages of a process which are least used, get stored in the secondary memory.

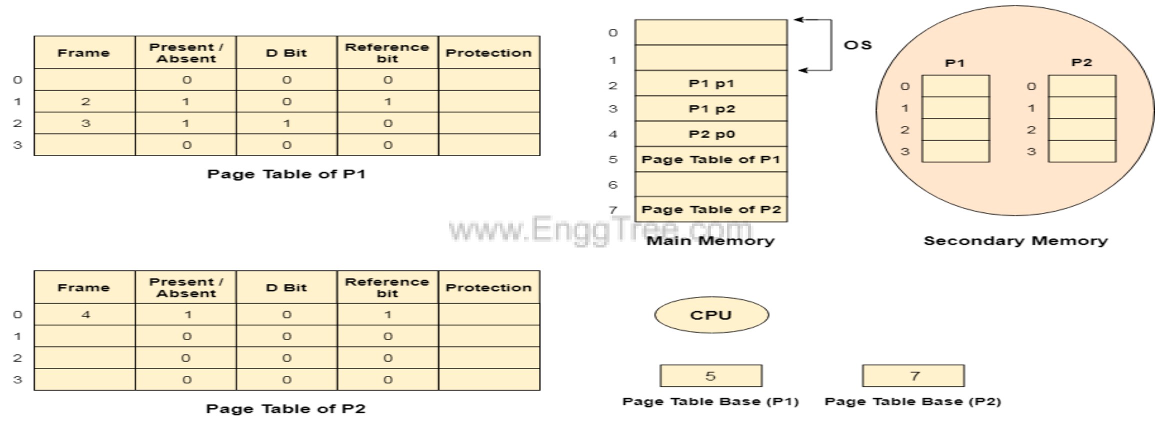
A page is copied to the main memory when its demand is made or page fault occurs. There are various page replacement algorithms which are used to determine the pages which will be replaced. We will discuss each one of them later in detail.

Snapshot of a virtual memory management system

Let us assume 2 processes, P1 and P2, contains 4 pages each. Each page size is 1 KB. The main memory contains 8 frame of 1 KB each. The OS resides in the first two partitions. In the third partition, 1st page of P1 is stored and the other frames are also shown as filled with the different pages of processes in the main memory.

The page tables of both the pages are 1 KB size each and therefore they can be fit in one frame each. The page tables of both the processes contain various information that is also shown in the image.

The CPU contains a register which contains the base address of page table that is 5 in the case of P1 and 7 in the case of P2. This page table base address will be added to the page number of the Logical address when it comes to accessing the actual corresponding entry.



	Frame	Present / Absent	D Bit	Reference bit	Protection
0		0	0	0	
1	2	1	0	1	
2	3	1	1	0	
3		0	0	0	

Page Table of P1

	Frame	Present / Absent	D Bit	Reference bit	Protection
0	4	1	0	1	
1		0	0	0	
2		0	0	0	
3		0	0	0	

Page Table of P2

Advantages of Virtual Memory

1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory

1. The system becomes slower since swapping takes time.
2. It takes more time in switching between applications.
3. The user will have the lesser hard disk space for its use.

DIRECT MEMORY ACCESS

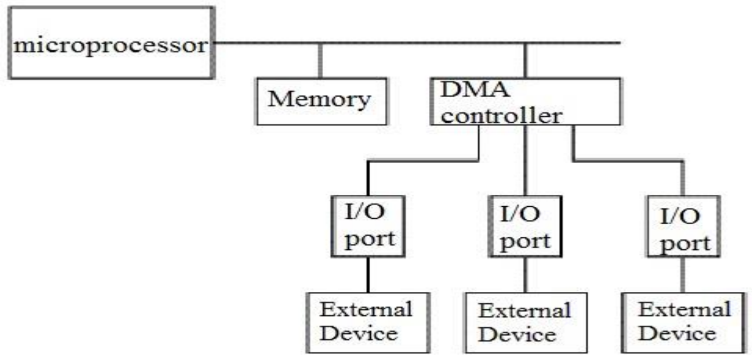
For the execution of a computer program, it requires the synchronous working of more than one component of a computer. For example, Processors – providing necessary control information, addresses…etc, buses – to transfer information and data to and from memory to I/O devices…etc. The interesting factor of the system would be the way it handles the transfer of information among processor, memory and I/O devices. Usually, processors control all the process of transferring data, right from initiating the transfer to the storage of data at the destination. This adds load on the processor and most of the time it stays in the ideal state, thus decreasing the efficiency of the system. To speed up the transfer of data between I/O devices and memory, DMA controller acts as station master. DMA controller transfers data with minimal intervention of the processor.

What is a DMA Controller?

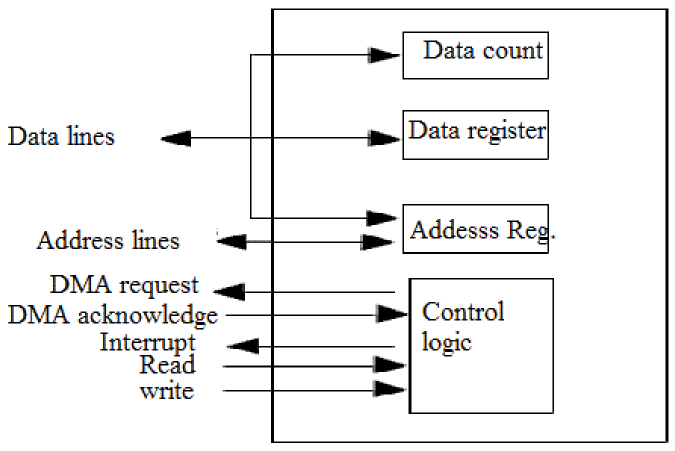
The term DMA stands for direct memory access. The hardware device used for direct memory access is called the DMA controller. DMA controller is a control unit, part of I/O device's interface circuit, which can transfer blocks of data between I/O devices and main memory with minimal intervention from the processor.

DMA Controller Diagram in Computer Architecture

DMA controller provides an interface between the bus and the input-output devices. Although it transfers data without intervention of processor, it is controlled by the processor. The processor initiates the DMA controller by sending the starting address, Number of words in the data block and direction of transfer of data .i.e. from I/O devices to the memory or from main memory to I/O devices. More than one external device can be connected to the DMA controller.



DMA controller contains an address unit, for generating addresses and selecting I/O device for transfer. It also contains the control unit and data count for keeping counts of the number of blocks transferred and indicating the direction of transfer of data. When the transfer is completed, DMA informs the processor by raising an interrupt. The typical block diagram of the DMA controller is shown in the figure below.



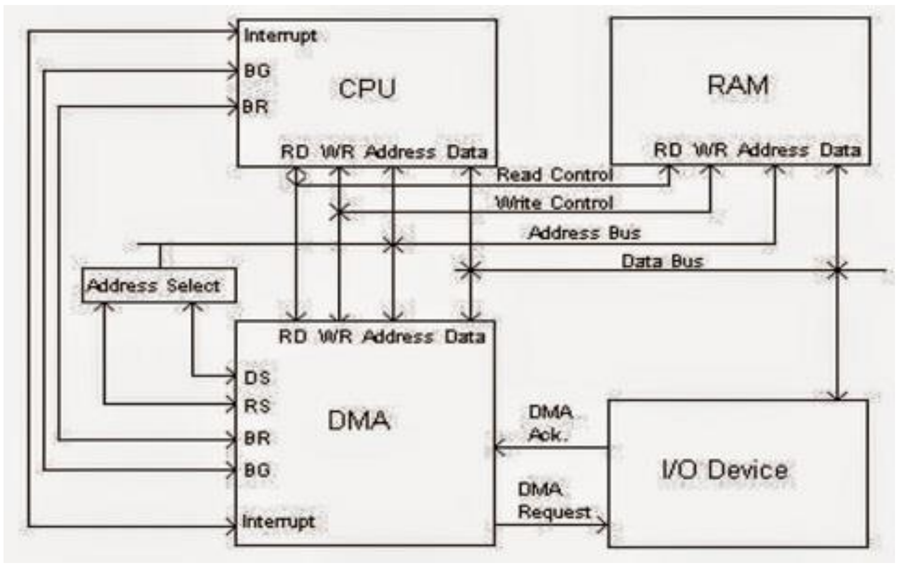
Working of DMA Controller
DMA controller has to share the bus with the processor to make the data transfer. The device that holds the bus at a given time is called bus master. When a transfer from I/O device to the memory or vice verse has to be made, the processor stops the execution of the current program, increments the program counter, moves data over stack then sends a DMA select signal to DMA controller over the address bus.
If the DMA controller is free, it requests the control of bus from the processor by raising the bus request signal. Processor grants the bus to the controller by raising the bus grant signal, now DMA controller is the bus master. The processor initiates the DMA controller by sending the memory addresses, number of blocks of data to be transferred and direction of data transfer. After assigning the data transfer task to the DMA controller, instead of waiting ideally till completion of data transfer, the processor resumes the execution of the program after retrieving instructions from the stack.

MA controller now has the full control of buses and can interact directly with memory and I/O devices independent of CPU. It makes the data transfer according to the control instructions received by the processor. After completion of data transfer, it disables the bus request signal and CPU disables the bus grant signal thereby moving control of buses to the CPU.

When an I/O device wants to initiate the transfer then it sends a DMA request signal to the DMA controller, for which the controller acknowledges if it is free. Then the controller requests the processor for the bus, raising the bus request signal. After receiving the bus grant signal it transfers the data from the device. For n channeled DMA controller n number of external devices can be connected.

The DMA transfers the data in three modes which include the following.

a) **Burst Mode**: In this mode DMA handover the buses to CPU only after completion of whole data transfer. Meanwhile, if the CPU requires the bus it has to stay ideal and wait for data transfer.
b) **Cycle Stealing Mode**: In this mode, DMA gives control of buses to CPU after transfer of every byte. It continuously issues a request for bus control, makes the transfer of one byte and returns the bus. By this CPU doesn't have to wait for a long time if it needs a bus for higher priority task.
c) **Transparent Mode:** Here, DMA transfers data only when CPU is executing the instruction which does not require the use of buses.



Advantages and Disadvantages of DMA Controller
The advantages and disadvantages of DMA controller include the following.

Advantages

- DMA speedups the memory operations by bypassing the involvement of the CPU.
- The work overload on the CPU decreases.
- For each transfer, only a few numbers of clock cycles are required

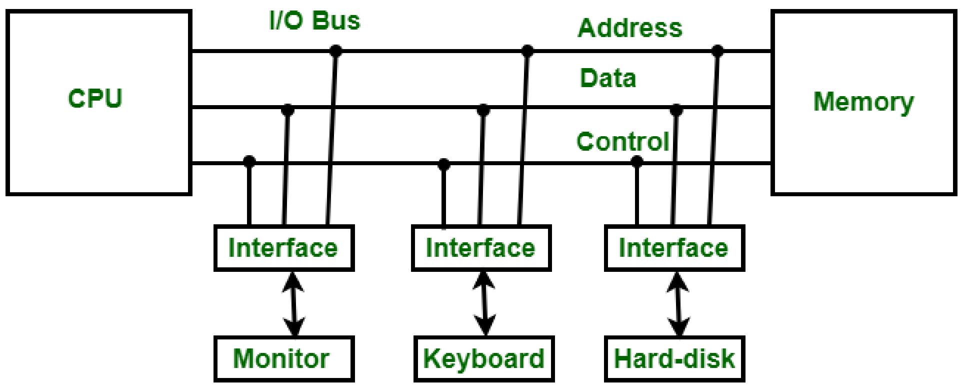
Disadvantages

- Cache coherence problem can be seen when DMA is used for data transfer.
- Increases the price of the system.

DMA (Direct Memory Access) controller is being used in graphics cards, network cards, sound cards etc… DMA is also used for intra-chip transfer in multi-core processors. Operating in one of its three modes, DMA can considerably reduce the load of the processor.

INTRODUCTION TO INPUT- OUTPUT INTERFACE:

Input -Output Interface is used as an method which helps in transferring of information between the internal storage devices i.e. memory and the external peripheral device . A peripheral device is that which provide input and output for the computer, it is also called Input-Output devices. For Example: A keyboard and mouse provide Input to the computer are called input devices while a monitor and printer that provide output to the computer are called output devices. Just like the external hard-drives, there is also availability of some peripheral devices which are able to provide both input and output.



Input-Output Interface

In micro-computer base system, the only purpose of peripheral devices is just to provide **special communication links** for the interfacing them with the CPU. To resolve the differences between peripheral devices and CPU, there is a special need for communication links.
The major differences are as follows:

1. The nature of peripheral devices is electromagnetic and electro-mechanical. The nature of the CPU is electronic. There is a lot of difference in the mode of operation of both peripheral devices and CPU.
2. There is also a synchronization mechanism because the data transfer rate of peripheral devices are slow than CPU.
3. In peripheral devices, data code and formats are differ from the format in the CPU and memory.
4. The operating mode of peripheral devices are different and each may be controlled so as not to disturb the operation of other peripheral devices connected to CPU.

There is a special need of the additional hardware to resolve the differences between CPU and peripheral devices to supervise and synchronize all input and output devices.

Functions of Input-Output Interface:

1. It is used to synchronize the operating speed of CPU with respect to input-output devices.
2. It selects the input-output device which is appropriate for the interpretation of the input-output device.
3. It is capable of providing signals like control and timing signals.
4. In this data buffering can be possible through data bus.
5. There are various error detectors.
6. It converts serial data into parallel data and vice-versa.
7. It also convert digital data into analog signal and vice-versa.

I/O Interface (Interrupt and DMA Mode)

The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode of Transfer:
The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.
2. Interrupt- initiated I/O.
3. Direct memory access(DMA).

Now let's discuss each mode one by one.

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.
   Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

Note: Both the methods programmed I/O and Interrupt-driven I/O require the active intervention of the
processor to transfer data between memory and the I/O module, and any data transfer must transverse
a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.

- The I/O transfer rate is limited by the speed with which the processor can test and service a
  device.
- The processor is tied up in managing an I/O transfer; a number of instructions must be executed
  for each I/O transfer.

3. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

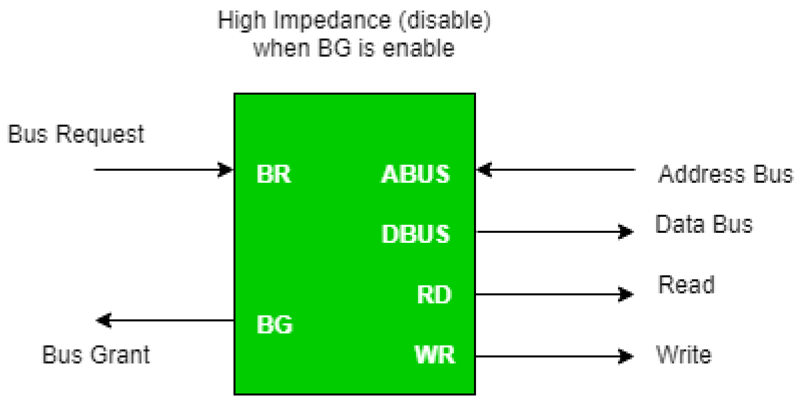


Figure - CPU Bus Signals for DMA Transfer

Bus Request : It is used by the DMA controller to request the CPU to relinquish the control of the buses.
Bus Grant : It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

Types of DMA transfer using DMA controller:
BurstTransfer :
DMA returns the bus after complete data transfer. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.
Steps involved are:

1. Bus grant request time.
2. Transfer the entire block of data at transfer rate of device because the device is usually slow than the speed at which the data can be transferred to CPU.
3. Release the control of the bus back to CPU
   So, total time taken to transfer the N bytes
   = Bus grant request time + (N) * (memory transfer rate) + Bus release control time.

Where,

X µsec =data transfer time or preparation time (words/block)

Y µsec =memory cycle time or cycle time or transfer time (words/block)

% CPU idle (Blocked)=(Y/X+Y)*100

% CPU Busy=(X/X+Y)*100

CyclicStealing :
An alternative method in which DMA controller transfers one word at a time after which it must return the control of the buses to the CPU. The CPU delays its operation only for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle.
Steps Involved are:

4. Buffer the byte into the buffer
5. Inform the CPU that the device has 1 byte to transfer (i.e. bus grant request)
6. Transfer the byte (at system bus speed)
7. Release the control of the bus back to CPU.

Before moving on transfer next byte of data, device performs step 1 again so that bus isn’t tied up and
the transfer won’t depend upon the transfer rate of device.
So, for 1 byte of transfer of data, time taken by using cycle stealing mode (T).
= time required for bus grant + 1 bus cycle to transfer data + time required to release the bus, it will be
N x T

In cycle stealing mode we always follow pipelining concept that when one byte is getting transferred then Device is parallel preparing the next byte. “The fraction of CPU time to the data transfer time” if asked then cycle stealing mode is used.

Where,

X µsec =data transfer time or preparation time

(words/block)

Y µsec =memory cycle time or cycle time or transfer

time (words/block)

% CPU idle (Blocked) =(Y/X)*100

% CPU busy=(X/Y)*100

Interleaved mode: In this technique , the DMA controller takes over the system bus when the
microprocessor is not using it.An alternate half cycle i.e. half cycle DMA + half cycle processor.

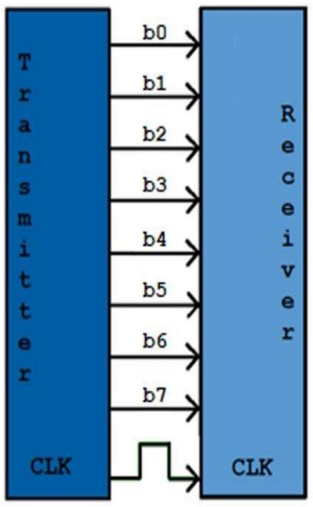
<u>ACCESSING I/O DEVICE</u>

We can access the i/o device in two interfaces.

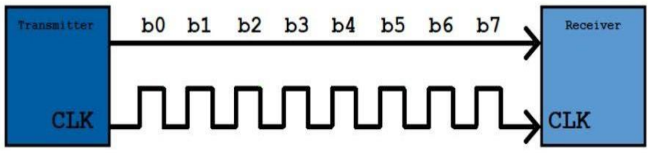
i. Serial interface
ii. Parallel interface

The main difference between the serial and parallel interfaces is how they transmit data. In serial interface the data is sent or received one bit at a time over a series of clock pulses.

In parallel mode the interface sends and receives 4 bits, 8 bits, or 16 bits of dataat a time over multiple transmission lines. These two interface modes will be explained in further detail below.



8-bit Parallel Interface



8-bit Serial Interface

Serial interface

Serial interface send a byte by bit by bit in serial manner with defined clockpulse width (with parity).

Serial interfaces consist of 3 types each with their own pins:

1. I2C (Inter-Integrated Circuit): Serial Data In and Serial Clock
2. 3/4-wire SPI (Serial Peripheral Interface): Consists of Serial Data Out, SerialData In, Serial Clock and an additional Chip Select pin for the 4-wire SPI
3. Serial synchronous control and data lines: Serial Data In, Register Select,Reset, and Serial Clock

Below is the serial interface connection example.

No.	Symbol	Description	Notes
1	SCL	Serial Clock	Output from master
2	CS	Chip Select, Low is active	Control Line
3	SDI/SDA	Serial Data In	Data lines
4	SDO	Serial Data Out	
5	A0	Register Select. 0: instruction, 1: data register	
6	RES	External Reset, Low is active	

Serial Interface Pros:

-Less data pins

-Cheaper

-Easy setup

Parallel Interface

The parallel interface transmits 8-bits, or one byte, of data over multiple data bus lines over one clock pulse. This makes parallel transmission faster than serial but istypically more expensive and requires more data pins to be connected. The parallelinterface consists of 8 data pins and 3 control pins. The control pins are typically labeled: Register Select (RS), Enable (E), and Read/Write (R/W). Additional common parallel interface pins may include: Contrast adjust (V0), Chip Select (CS) and

Parallel interface consists of 2 standard types:

1. 8080 type: parallel 4-bit/8-bit data input with a write and a read line
2. 6800 type: parallel 4/8-bit data input with write, read and enable lines

Parallel Interface Pros

-Faster data transmission
-High performance

INTERCONNECTION STANDARDS

There are two interconnection standards

i. USB
ii. SATA (Serial ATA)

Universal Serial Bus (USB)

The **universal serial bus (USB)** is a standard interface for connecting a wide range of devices to the computer such as keyboard, mouse, smartphones, speakers, cameras etc. The USB was introduced for commercial use in the year 1995 at that time it has a data transfer speed of 12 megabits/s.

With some improvement, a modified USB 2 was introduced which is also called a *highspeed USB* that transfers data at 480 megabits/s. With the evolution of I/O devices that require highspeed data transfer also leads to the development of USB 3 which is also referred to as *Superspeed USB* which transfers data at 5 gigabits/s. The recent version of USB can transfer data up to 20 gigabits/s.

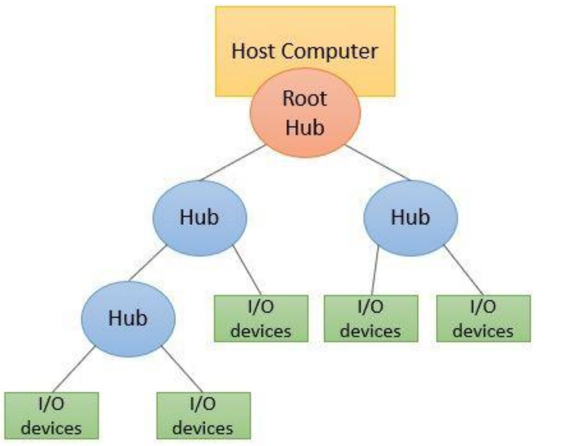
Key Objectives of Universal Serial Bus

- The developed USB must be *simple* and a *low-cost* interconnection system that should be *easy* to use.
- The developed USB must be compatible with all new I/O devices, their bit rates, internet connections and audio, video application.
- The USB must support a *plug-and-play* mode of operation.
- The USB must support *low power* implementation.
- The USB must also provide support for *legacy hardware* and *software*.

USB Architecture

When multiple I/O devices are connected to the computer through USB they all are organized in a tree structure. Each I/O device makes a *point-to-point* connection and transfers data using the *serial transmission format* we have discussed serial transmission in our previous content ‘interface circuit’.

A tree structure has a **root, nodes** and **leaves.** The tree structure connecting I/O devices to the computer using USB has nodes which are also referred to as a **hub**. Hub is the intermediatory connecting point between the I/O devices and the computer. Every tree has a root here, it is referred to as the **root hub** which connects the entire tree to the hosting computer. The leaves of the tree here are nothing but the I/O devices such as a mouse, keyboard, camera, speaker.



Universal Serial Bus Tree Structure

The USB works on the principle of polling. In **polling**, the processor keeps on checking whether the I/O device is ready for data transfer or not. So, the devices do not have to inform the processor about any of their statuses. It is the processor's responsibility to keep a check. This makes the USB simple and low cost.

Whenever a new device is connected to the hub it is addressed as 0. Now at a regular interval the host computer polls all the hubs to get their status which lets the host know of I/O devices that are either detached from the system or are attached to the system.

When the host becomes aware of the new device it gets to know about the capabilities of the device by reading the information present in the special memory of the device's USB interface. So that the host can use the appropriate device driver to communicate with the device.

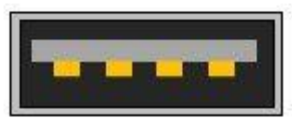
The host then assigns an address to this new device, this address is written to the register of the device interface register. With this mechanism, USB serves plug-and-play capability.

The **plug and play** feature let the host recognize the existence of the new I/O device automatically when the device is plugged in. The host software determines the capabilities of the I/O devices and if it has any special requirement.

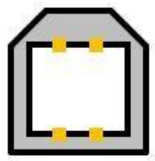
The USB is **hot-pluggable** which means the I/O device can be attached or removed from the host system without performing any restart or shutdown. That means your system can keep running while the I/O device is plugged or removed.
Types of USB Connectors

The USB has different types of ports and connectors. Usually, the upstream port and connector are always the USB type A the downstream port and connector differ depending on the type of device connected. We will discuss all types of the USB connector.

USB Type A: This is the standard connector that can be found at one end of the USB cable and is also known as upstream. It has a flat structure and has four connecting lines as you can see in the image below.


USB Type A

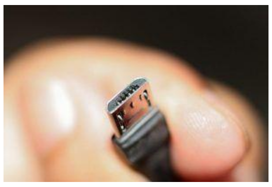
USB Type B: This is an older standard cable and was used to connect the peripheral devices also referred to as downstream. It is approximately a square as you can see in the image below. This is now been replaced by the newer versions.


USB Type B

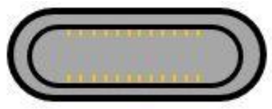
Mini USB: This type of USB is compatible with mobile devices. This type of USB is now superseded your micro-USB still you will get it on some devices.

Mini USB

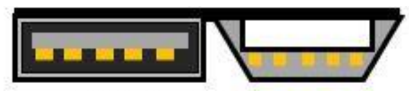
Micro USB: This type of USB is found on newer mobile devices. It has a compact 5 pin design.



USB Type C: This type of USB is used for transferring both data and power to the attached peripheral or I/O device. The USB C does not have a fixed orientation as it is reversible i.e. you can plug it upside down or in reverse.



USB 3.0 Micro B: This USB is a superspeed USB. This USB is used for a device that requires high-speed data transfer. You can find this kind of USB on portable hard drives.



Electrical Characteristics of USB

The standard USB has four lines of connection among which two carry power (one carry +5 V and one is for Ground). The other two lines of connection are for data transfer. USB also supply power to connected I/O device that requires very low power.

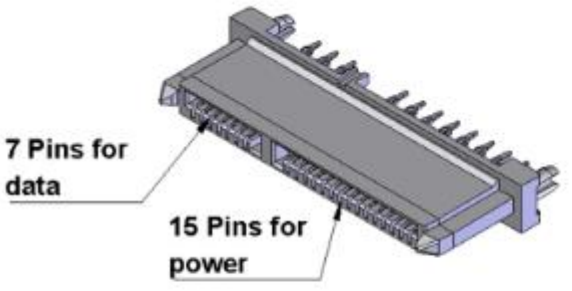
Transferring of data over USB can be divided into two categories i.e., transferring data at low speed and transferring data at high speed.

The low-speed transmission uses **single-ended signalling** where varying high voltage is transmitted over one of the two data lines to represent the signal bit 0 or 1. The other data line is connected to the reference voltage i.e., ground. The single-ended signalling is prone to noise.

The high-speed data transmission uses the approach **differential signalling**. Here, the signal is transmitted over the two data lines that are twisted together. Here both the data lines are involved in carrying the signal no ground wire is required. The differential signalling is not prone to noise and uses low voltages as compared to single-ended transmission.

SATA

SATA stands for **Serial Advanced Technology Attachment or Serial ATA**.
SATA is an **interface that connects various storage devices such as hard disks, optical drives, SSD's, etc to the motherboard**. SATA was introduced in the year 2000 to replace the long-standing PATA (Parallel ATA) interface. We all know, in serial mode, data is transferred bit by bit and in parallel, there are several streams that carry the data. Despite knowing this fact, there is a drawback in PATA. PATA is highly susceptible to outside interferences and hence allows SATA to operate at high speeds than PATA. SATA cables are thinner, more flexible and compact as compared to the PATA cables.
There were several industry groups that began their development in SATA late in the 2000s. It was only in the year 2003 that SATA-IO (SATA International Organization) was formed and it laid out the first SATA specifications.



A SATA controller is a device that is used to connect the computer's motherboard to the storage drives.

SATA operates on two modes –

1. **IDE mode:** IDE stands for Integrated Drive Electronics. This is a mode which is used to provide backward compatibility with older hardware, which runs on PATA, at the cost of low performance.
2. **AHCI mode:** AHCI is abbreviation for Advanced Host Controller Interface. AHCI is a high-performance mode that also provides support for hot-swapping.

Characteristics of SATA

- **Low Voltage Requirement:** SATA operates on 500mV (0.5V) peak-to-peak signaling. This help in promoting a much low interference and crosstalk between conductors.
- **Simplified construction:** PATA cables had 40-pin/80-wire ribbon cable. This was complex in construction. In comparison, SATA had a single 7 pin data cable and a 15 pin power cable. This cable resulted in a higher signaling rate, which translates to faster throughput of data.
- **Differential Signaling:** SATA uses differential signaling. Differential signaling is a technology which uses two adjacent wires to simultaneously the in-phase and out-of-phase signals. Thus, it is possible to transfer high-speed data with low operating voltage and low power consumption by detecting the phase difference between the two signals at the receiver's end.
- **High data transfer rate:** SATA has a high data transfer rate of 150/300/600 MBs/second. This capability of SATA allows for faster program loading, better picture loading and fast document loading.

Advantages of SATA

- Faster data transfer rate as compared to PATA.
- SATA cable can be of length upto 1 meter, whereas PATA cable can only have length of maximum 18 inches.
- SATA cables are smaller in size.
- Since, they are smaller in size, they take up less space inside the computer and increase the internal air flow. Increased air flow can decrease heat build-up and therefore increases the overall life of computer.
- Most modern computer motherboards today have SATA ports more than PATA ports.
- Low power consumption (0.5V).

Disadvantages of SATA

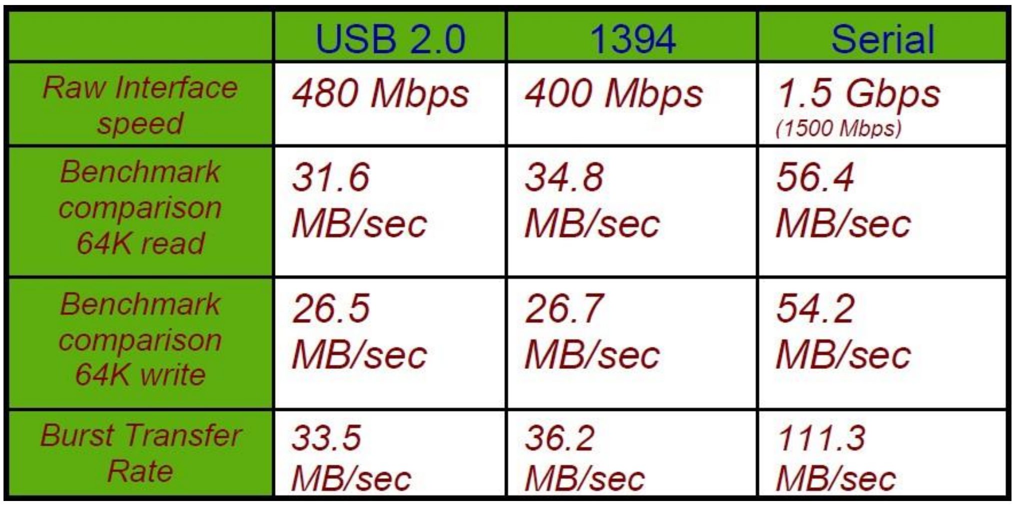
- Special device drivers are required sometimes to recognize and use the drive. However, a SATA hard drive can behave as a PATA
  drive. This eliminates the need for a specific driver to be installed.
- SATA cable supports only one hard drive to connect at a time, whereas PATA cable allows up to two PATA drives per cable.
- SATA is costlier as compared to PATA.

SATA standards and revisions

The nonprofit SATA-IO industry consortium authors the technical specifications governing Serial ATA device interfaces. The consortium revises SATA standards to reflect increased data transfer rates. These revisions include the following changes:

- **SATA Revision 1.** These devices were widely used in personal desktop and office computers, configured from PATA drives daisy chained together in a primary/secondary configuration. SATA Revision 1 devices reached a top transfer rate of 1.5 Gbps.
- **SATA Revision 2.** These devices doubled the transfer speed to 3.2 Gbps with the inclusion of port multipliers, port selectors and improved queue depth.
- **SATA Revision 3.** These interfaces supported drive transfer rates up to 6 Gbps. Revision 3 drives are backward-compatible with SATA Revision 1 and Revision 2 devices, though with lower transfer speeds.
- **SATA Revision 3.1.** This intermediate revision added final design requirements for SATA Universal Storage Module for consumer-based portable storage applications.
- **SATA Revision 3.2.** This update added the SATA Express specification. It supports the simultaneous use of SATA ports and PCI Express (PCIe) lanes.
- **SATA Revision 3.3.** This revision addressed the use of shingled magnetic recording
- **SATA Revision 3.5.** This change promoted greater integration and interoperability with PCIe flash and other I/O protocols.

COMPARISON:



	USB 2.0	1394	Serial
Raw Interface speed	*480 Mbps*	*400 Mbps*	*1.5 Gbps (1500 Mbps)*
Benchmark comparison 64K read	*31.6 MB/sec*	*34.8 MB/sec*	*56.4 MB/sec*
Benchmark comparison 64K write	*26.5 MB/sec*	*26.7 MB/sec*	*54.2 MB/sec*
Burst Transfer Rate	*33.5 MB/sec*	*36.2 MB/sec*	*111.3 MB/sec*