K.RAMAKRISHNAN COLLEGE OF ENGINEERING (AUTONOMOUS)

SAMAYAPURAM,TRICHY–621112.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
(ACCREDITED BY NBA)

NAME :

ROLL NO :

BATCH :

YEAR/SEC :

SUB CODE : UEC1511

SUBJECT : Digital Signal Processing Laboratory

INDEX

Ex. No : 1	GENERATION OF ELEMENTARY DISCRETE - TIME SEQUENCES
Date :	

AIM:

To write a MATLAB program to generate various discrete- time sequences.

SOFTWARE REQUIRED:

MATLAB (R2020) software.

ALGORITHM:

UNIT IMPULSE RESPONSE:

- Start the program
- Enter the value of N
- Generates zeros and ones for the corresponding values of n to get the impulse response
- Plot the output.

UNIT STEP RESPONSE:

- Start the program
- Enter the value of N
- Generates ones for the corresponding values of n to get the unit step response
- Plot the output.

EXPONENTIAL RESPONSE

- Start the program
- Enter the value of N
- Generates the corresponding values of n to get the exponential response
- Plot the output.

UNITRAMPRESPONSE:

- Start the program
- Enter the value of N
- Generates the corresponding values of n to get the unit ramp response
- Plot the output.

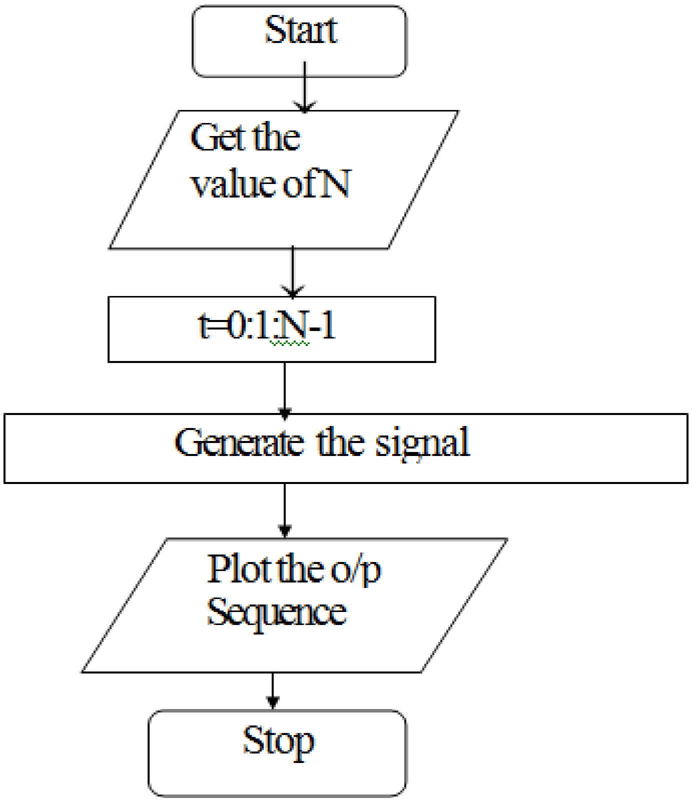
SINEWAVE:

- Start the program
- Enter the value of N
- Generates the corresponding values of n to get the sine wave using sin function
- Plot the output.

COSINE WAVE:

- Start the program
- Enter the value of N
- Generates the corresponding values of n to get the cosine wave using cos function
- Plot the output.

FLOWCHART:



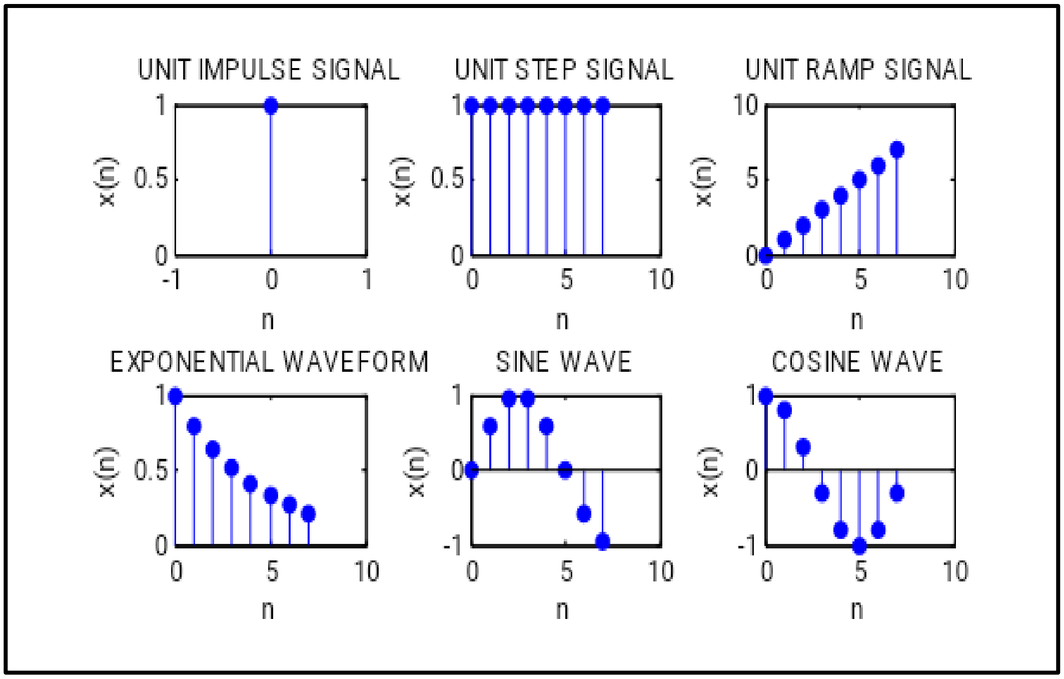
GENERATION OF ELEMENTARY DISCRETE - TIME SEQUENCES

PROGRAM:

```
clc;
clear all;
close all;

%UNIT IMPULSE SIGNAL%
N=8;
n=0;
x=ones(1,1);
subplot(3,3,1);
stem(n,x);
xlabel('n');
ylabel('x(n)');
title('UNIT IMPULSE SIGNAL');

%UNIT STEP SIGNAL%
N=8;
n=0:1:N-1;
x=ones(1,N);
subplot(3,3,2);
stem(n,x);
xlabel('n');
ylabel('x(n)');
title('UNIT STEP SIGNAL');

%UNIT RAMP SIGNAL%
N=8;
n=0:1:N-1;
x=0:1:N-1;
subplot(3,3,3);
stem(n,x);
xlabel('n');
ylabel('x(n)');
title('UNIT RAMP SIGNAL');

%EXPONENTIAL WAVE%
N=8;
n=0:1:N-1;
x=exp(n);
subplot(3,3,4);
stem(n,x );
xlabel('n');
ylabel('x(n)');
title('EXPONENTIAL WAVEFORM');
```

```
%SINE WAVE%
N=8;
n=0:1:N-1;
x=sin(.2*pi*n);
subplot(3,3,5);
stem(n,x);
xlabel('n');
ylabel('x(n)');
title('SINE WAVE');

%COSINE WAVE%
N=8;
n=0:1:N-1;
x=cos(.2*pi*n);
subplot(3,3,6);
stem(n,x);
xlabel('n');
ylabel('x(n)');
title('COSINE WAVE');
```

OUTPUT:



RESULT:

Thus the generation of various discrete time sequences has been performed using MATLAB program.

Ex. No : 2.A	**LINEAR CONVOLUTION USING CONVOLUTION FUNCTION**
Date :	

AIM:

To perform the linear convolution of two sequences using convolution function.

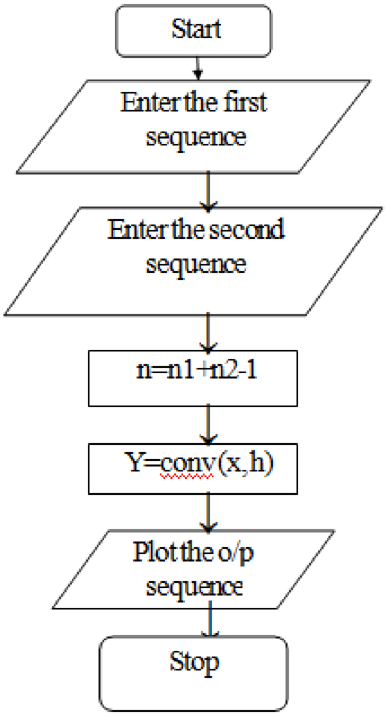
SOFTWARE REQUIRED:

MATLAB (R2020) software.

ALGORITHM:

- Start the program
- Enter the two input sequences x(n) and h(n)
- Find the length of the input and output sequences
- Find the convolution of the two sequences
- Display the output
- Plot the input sequences and the output sequence
- Stop the program

FLOWCHART:

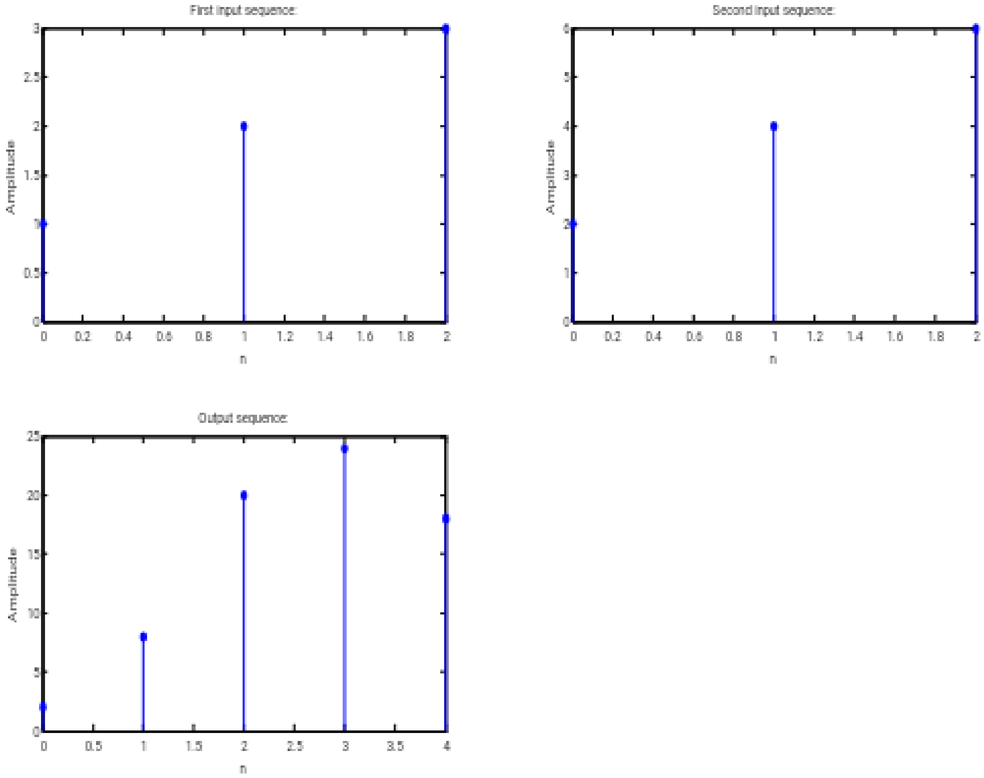


PROGRAM:

```
clc;
clear all;
close all;
x=input('Enter the first input sequence x(n)');
h=input('Enter the second input sequence h(n)');
n1=length(x);
n2=length(h);
n=n1+n2-1;
y=conv(x,h);
disp('Linear Convolution Output is:');
disp(y);
t1=0:n1-1;
subplot(2,2,1);
stem(t1,x);
xlabel('n');
ylabel('Amplitude');
title('First input sequence:');
t2=0:n2-1;
subplot(2,2,2);
stem(t2,h);
xlabel('n');
ylabel('Amplitude');
title('Second input sequence:');
t=0:1:n-1;
subplot(2,2,3);
stem(t,y);
xlabel('n');
ylabel('Amplitude');
title('Output sequence:');
```

OUTPUT:

```
Enter the first input sequence x(n) [1 2 3 ]
Enter the second input sequence h(n) [2 4 6]
Linear Convolution Output is :
     2     8    20    24    18
```



RESULT:

Thus the linear convolution of two sequences using convolution function was performed and executed successfully.

Ex. No : 2B	**LINEAR CONVOLUTION USING FFT**
Date :	

AIM:

To perform the linear convolution of two sequences using FFT.

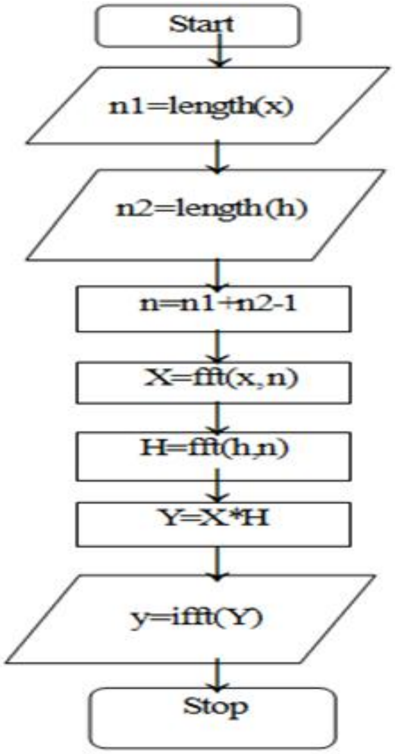
SOFTWARE REQUIRED:

MATLAB (R2020) software.

ALGORITHM:

- Start the program
- Enter the two input sequences x(n) and h(n)
- Find the length of the input and output sequences
- Find FFT of x(n) & h(n) to get X(K) & H(K)
- Multiply X(K) & H(K) to get Y(K)
- Find IFFT of Y(K) to get convolution output y(n).
- Display the output
- Plot the input & output sequences
- Stop the program

FLOWCHART :

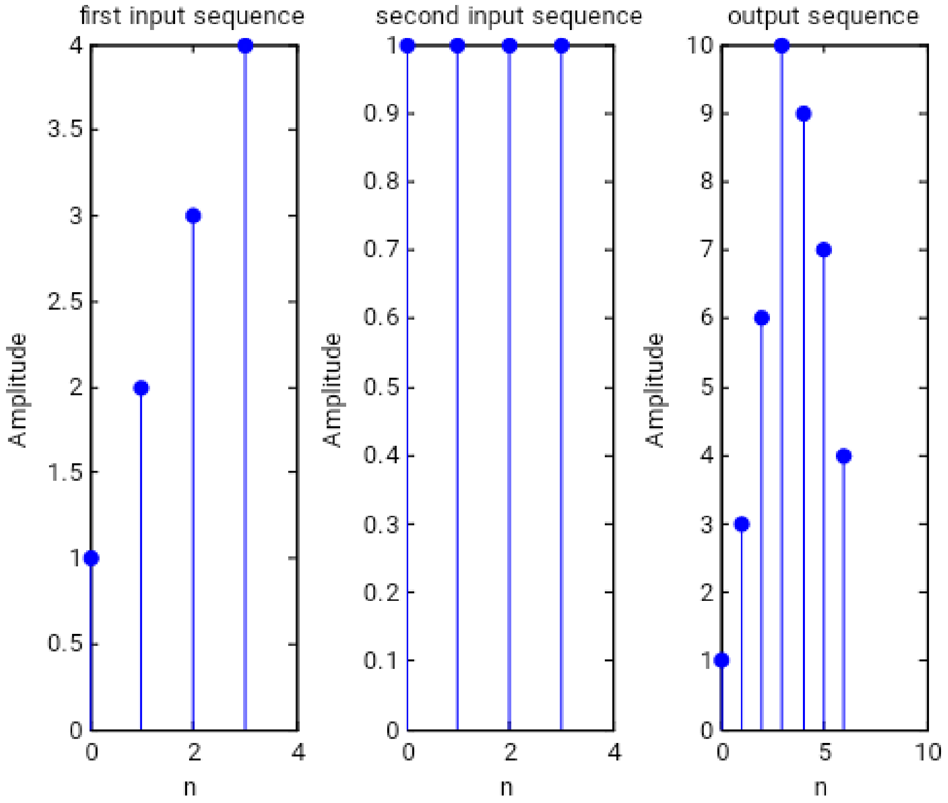


PROGRAM:

```
clc;
clear all;
close all;
x=input('Enter the first input sequence x(n):');
h=input('Enter the second input sequence h(n):');
n1=length(x);
n2=length(h);
n=n1+n2-1;
X=fft(x,n);
H=fft(h,n);
Y=X.*H;
y=ifft(Y);
disp('linear convolution output is:');
disp(y);
t1=0:n1-1;
subplot(1,3,1);
stem(t1,x);
xlabel('n');
ylabel('Amplitude');
title('first input sequence');
t2=0:n2-1;
subplot(1,3,2);
stem(t2,h);
xlabel('n');
ylabel('Amplitude');
title('second input sequence');
t=0:1:n-1;
subplot(1,3,3);
stem(t,y);
xlabel('n');
ylabel('Amplitude');
title('output sequence');
```

OUTPUT:

```
Enter the first input sequence x(n):[1 2 3 4]
Enter the second input sequence h(n):[1 1 1 1]
linear convolution output is:
    1.0000    3.0000    6.0000   10.0000    9.0000    7.0000    4.0000
```



RESULT:

Thus the linear convolution of two sequences using FFT was performed and executed using MATLAB.

Ex. No : 2. C	**CIRCULAR CONVOLUTION USING FFT**
Date :	

AIM:

To perform the circular convolution of two sequences using FFT.

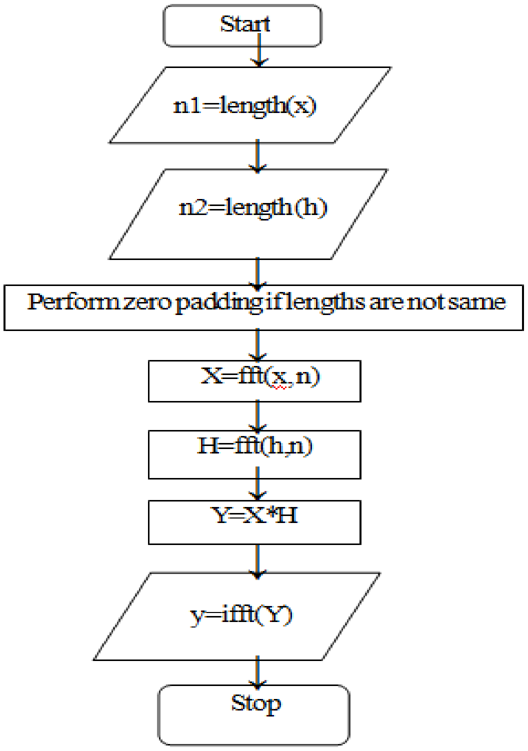
SOFTWARE REQUIRED:

MATLAB (R2020) software.

ALGORITHM:

- Start the program
- Enter the two input sequences x(n) and h(n)
- Find the length of the input and output sequences
- If the length of sequences are not same , perform zero padding to the sequence with less length
- Find FFT of x(n) & h(n) to get X(K) & H(K)
- Multiply X(K) & H(K) to get Y(K)
- Find IFFT of Y(K) to get convolution output y(n).
- Display the output
- Plot the input & output sequences
- Stop the program

FLOWCHART:

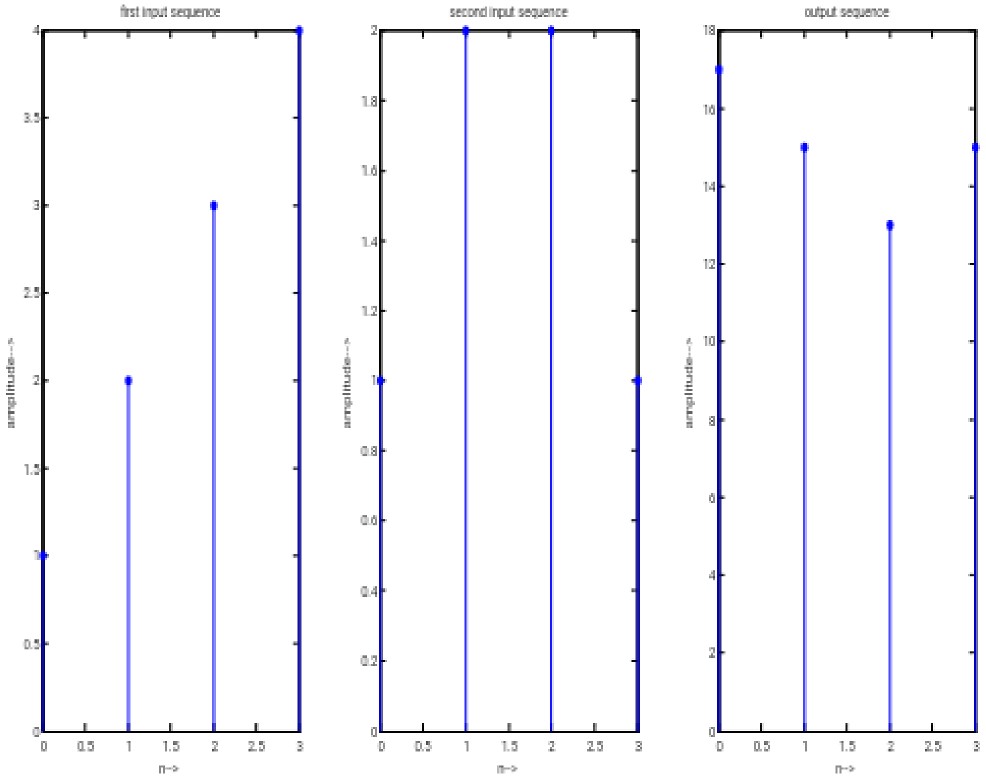


PROGRAM

```
clc;
clear all;
close all;
x1=input('enter the 1st input sequence');
x2=input('enter the 2nd input sequence');
n1=length(x1);
n2=length(x2);
if(n1<n2)
   x1=[zeros(1,n2-n1)];
elseif(n2<n1)
   x2=[zeros(1,n1-n2)];
else
   x1=x1;
   x2=x2;
end;
n1=length(x1);
n2=length(x2);
A=fft(x1,n1);
B=fft(x2,n2);
Y=A.*B;
y=ifft(Y);
n=length(y);
disp('circular convolution output is:');
disp(y);
t1=0:n1-1;
subplot(1,3,1);
stem(t1,x1);
xlabel('n-->');
ylabel('amplitude-->');
title('first input sequence');
t2=0:n2-1;
subplot(1,3,2);
stem(t2,x2);
xlabel('n-->');
ylabel('amplitude-->');
title('second input sequence');
t=0:1:n-1;
subplot(1,3,3);
stem(t,y);
xlabel('n-->');
ylabel('amplitude-->');
title('output sequence');
```

OUTPUT

```
Enter the 1st input sequence [1 2 3 4]
Enter the 2nd input sequence [1 2 2 1]
circular convolution output is:
17   15   13   15
```



RESULT:

Thus the circular convolution of two sequences using FFT was performed and executed using MATLAB.

Ex. No : 3	**AUTO CORRELATION AND CROSS CORRELATION**
Date :	

AIM:

To write a MATLAB program to perform autocorrelation and cross correlation.

SOFTWARE REQUIRED:

MATLAB (R2020) software.

ALGORITHM:

AUTO CORRELATION:

- Start the program
- Enter the value of sequence
- Generate the corresponding values of y to get the convolution using conv function
- Plot the output

CROSS CORRELATION:

- Start the program
- Enter the value of sequence
- Generate the corresponding values of y to get the convolution using conv function
- Plot the output

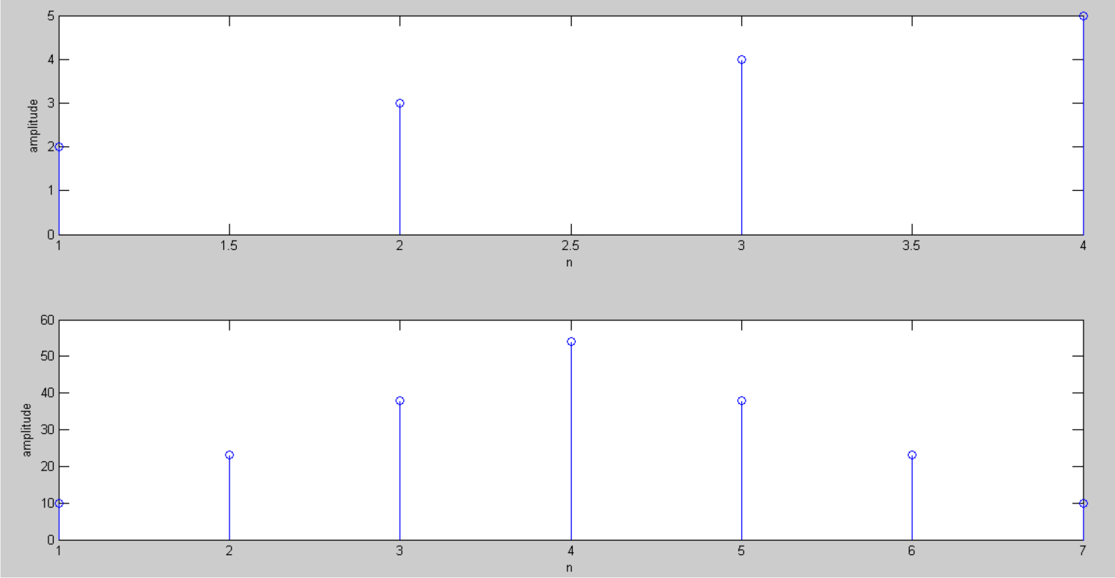
PROGRAM:

```
%AUTOCORRELATION%
clc;
clear all;
close all;
x=input('Enter the sequence');
y=xcorr(x,x);
figure;
subplot(2,1,1);
stem(x);
ylabel('amplitude');
xlabel('n');
subplot(2,1,2);
stem(fliplr(y));
y
xlabel('n');
ylabel('amplitude');
disp('the resultant signal is  ');
fliplr(y);


%CROSS CORRELATION%
clc;
close all;
clear all;
x=input('Enter the first sequence');
h=input('Enter the second sequence');
y=xcorr(x,h);
figure;
subplot(3,1,1);
stem(x);
xlabel('n');
ylabel('amplitude');
subplot(3,1,2);
stem(h);
ylabel('amplitude');
xlabel('n');
subplot(3,1,3);
stem(fliplr(y));
y
ylabel('amplitude');
xlabel('n');
title('The resultant signal is ');
fliplr(y)
```

OUTPUT:

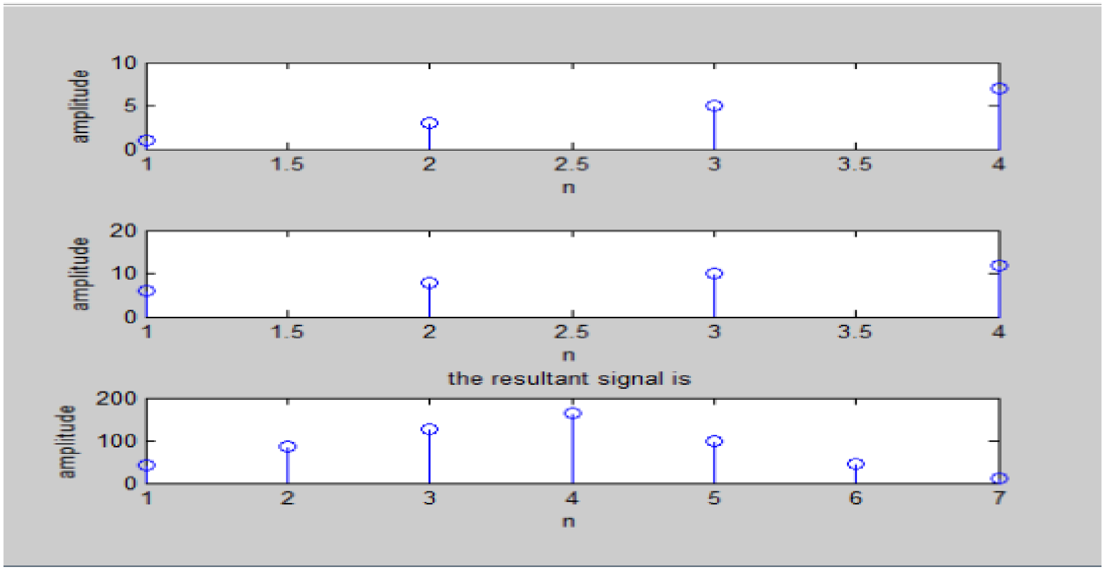
AUTO CORRELATION:

```
Enter the sequence [2 3 4 5]
The resultant signal is
y =   10.0000   23.0000   38.0000   54.0000   38.0000   23.0000   10.0000
```



CROSS CORRELATION:

```
enter the first sequence [1 2 3 4]
enter the second sequence [2 4 6 8]
y =8.0000   22.0000   40.0000   60.0000   40.0000   22.0000    8.0000
```



RESULT:

Thus the auto-correlation and cross-correlation between the sequences are performed and executed using MATLAB

Ex. No : 4	**FREQUENCY ANALYSIS OF DFT**
Date :	

AIM:

To perform the frequency analysis of DFT for the given sequence using FFT.

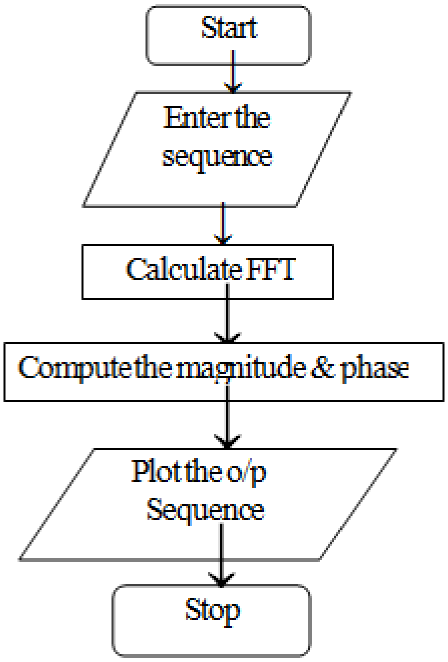
SOFTWARE REQUIRED:

MATLAB (R2020) software.

ALGORITHM:

- Start the program
- Enter the input sequences x(n)
- Find the length of the input sequence
- Find FFT of x(n) to get X(K) using the command fft(x,n)
- Find the magnitude response and phase response of X(K)
- Display the output
- Plot the input & output sequences (magnitude and phase response)
- Stop the program

FLOW CHART:

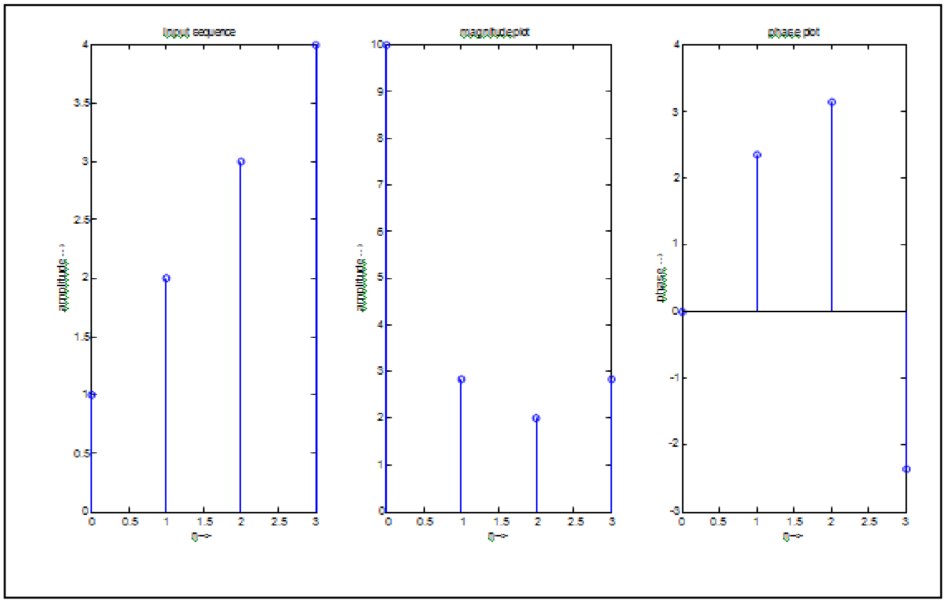


PROGRAM:

```
clc;
clear  all;
x=input ('Enter the sequence :');
n=length (x);
y=fft(x,n);
di sp('output  sequence is:');
disp(y);
m=abs(y);
disp('magnitude function is:');
disp(m);
p=angle(y);
disp('phase  function  is:’);
disp(p);
subplot(1,3,1);
t1=0 : n-1;
stem(t1,x);
xlabel('n-->');
ylabel('amplitude -->');
title('input sequence');
subplot(1,3,2);
t2=0 : n-1;
stem (t2,m);
xlabel('n-->');
ylabel('amplitude -->');
title('magnitude plot');
subplot(1,3,3);
t3=0 : n-1;
stem(t3,p);
xlabel('n-->');
ylabel('phase -->');
title('phase  plot');
```

OUTPUT:

```
Enter the sequence: [1 2 3 4]

output sequence is:
10.0000          -2.0000 + 2.0000i        -2.0000          -2.0000- 2.0000i

magnitude function is:
10.0000      2.8284   2.0000   2.8284

phase function is:
0   2.3562   3.1416        -2.3562
```



RESULT:

Thus the frequency analysis of DFT computation using FFT was performed and executed using MATLAB.

Ex. No : 5. A	FIR LOW PASS FILTER USING RECTANGULAR WINDOW
Date :	

AIM:

To design a FIR Low pass filter using rectangular window and to plot the frequency response.

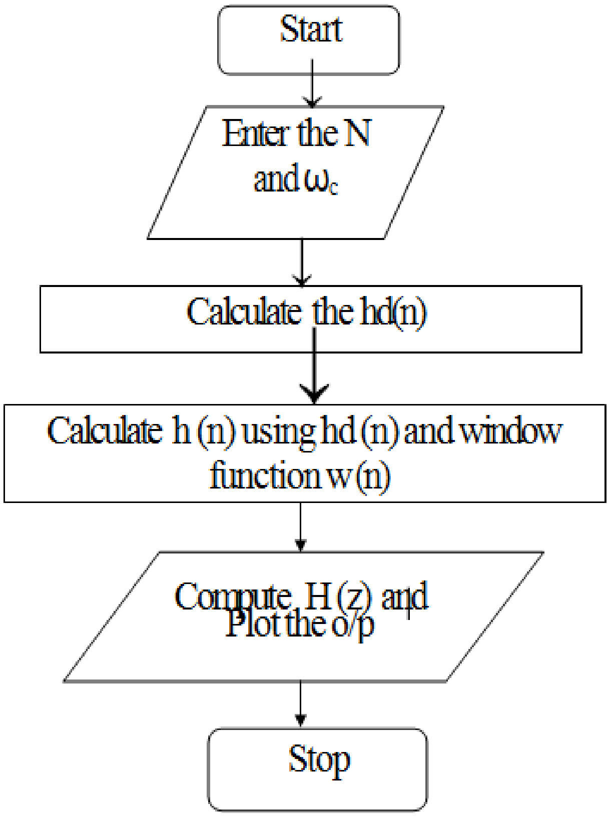
SOFTWARE REQUIRED:

MATLAB (R2020) software.

ALGORITHM:

- Start the program
- Get the cut-off frequency& order of the filter
- For the given specifications, find the desired impulse response hd(n)
- Find the corresponding window coefficients w(n) for the given order of the filter
- Find the filter coefficients & transfer function for the given specifications
- Obtain the magnitude response & phase response
- Plot the magnitude response and phase response
- Stop the program

FLOWCHART:

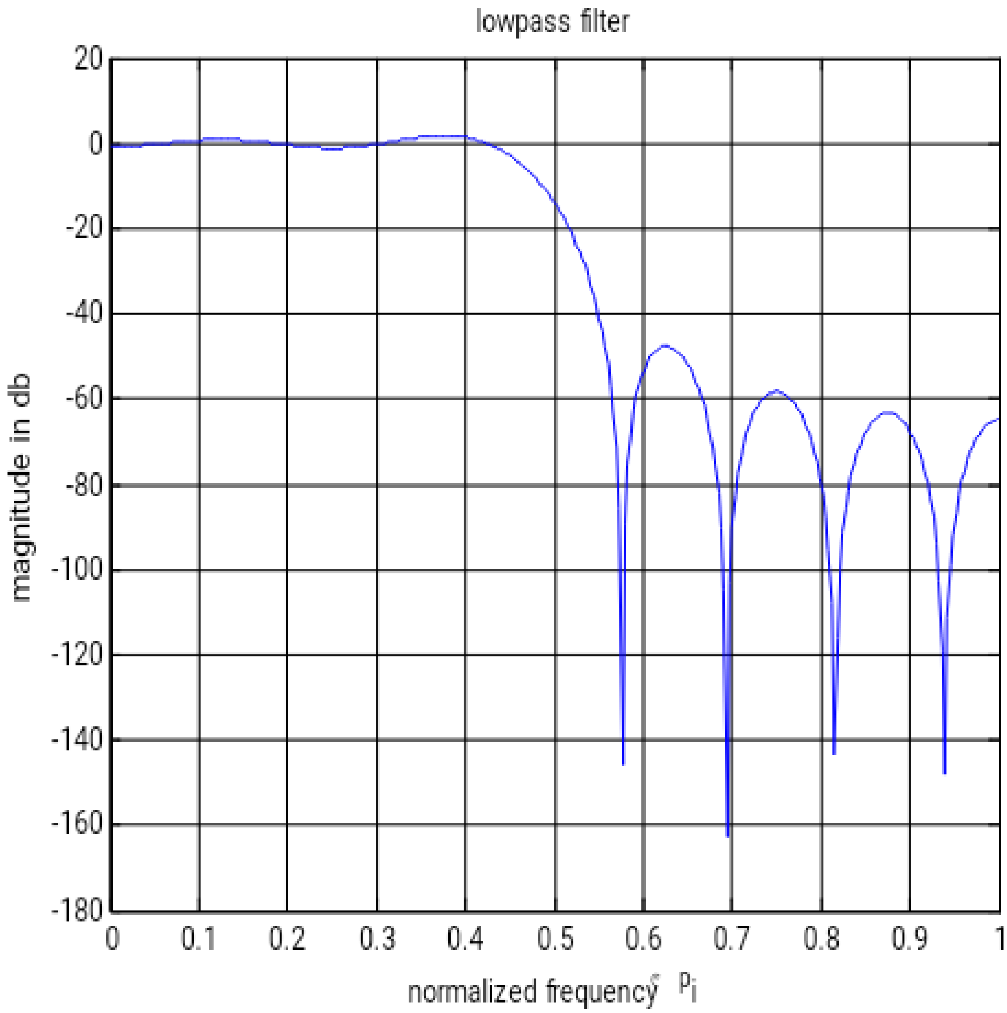

Start
Enter the N and ω_c
Calculate the hd(n)
Calculate h (n) using hd (n) and window function w(n)
Compute H (z) and Plot the o/p
Stop

PROGRAM:

```
clc;
clear all;
close all;
N=input('enter the order of filter');
wc=(pi/2);
a=(N-1)/2;
for n=1:N
    if((n-1)==a)
        hd(n)=(wc/pi);
    else
        hd(n)=(sin(wc*(n-1-a)))/(pi*(n-1-a));
    end;
    w(n)=1;
end;
h=w.*hd;
a=0:0.01:pi;
b=freqz(h,1,a);
mag=20*log(abs(b));
plot(a/pi,mag);
grid;
xlabel('normalized frequency\omega^pi');
ylabel('magnitude in db');
title('low pass filter');
```

OUTPUT:

```
Enter the order of filter15
```



RESULT:

Thus the FIR low pass filter using rectangular window was designed and executed using MATLAB.

Ex.no: 5.B	**FIR HIGH PASS FILTERUSING HAMMING WINDOW**
Date:	

AIM:

To design a FIR high pass filter using hamming window and to plot the frequency response.

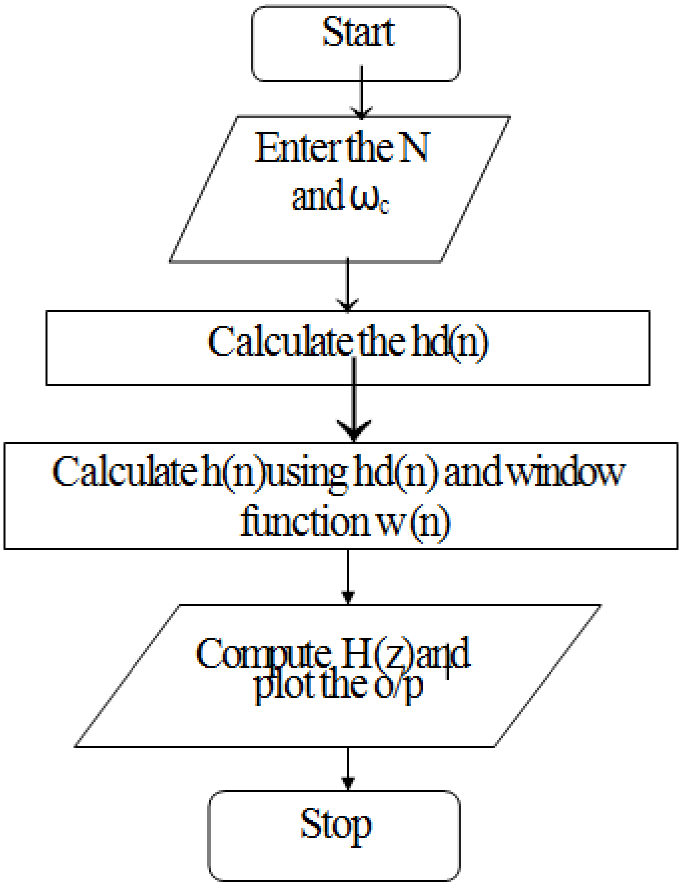
SOFTWARE REQUIRED:

MATLAB (R2020) Software.

ALGORITHM:

- Start the program
- Get the cut-off frequency& order of the filter
- For the given specifications, find the desired impulse response hd(n)
- Find the corresponding window coefficients w(n) for the given order of the filter
- Find the filter coefficients & transfer function for the given specifications
- Obtain the magnitude response & phase response
- Plot the magnitude response and phase response
- Stop the program
-

FLOWCHART:

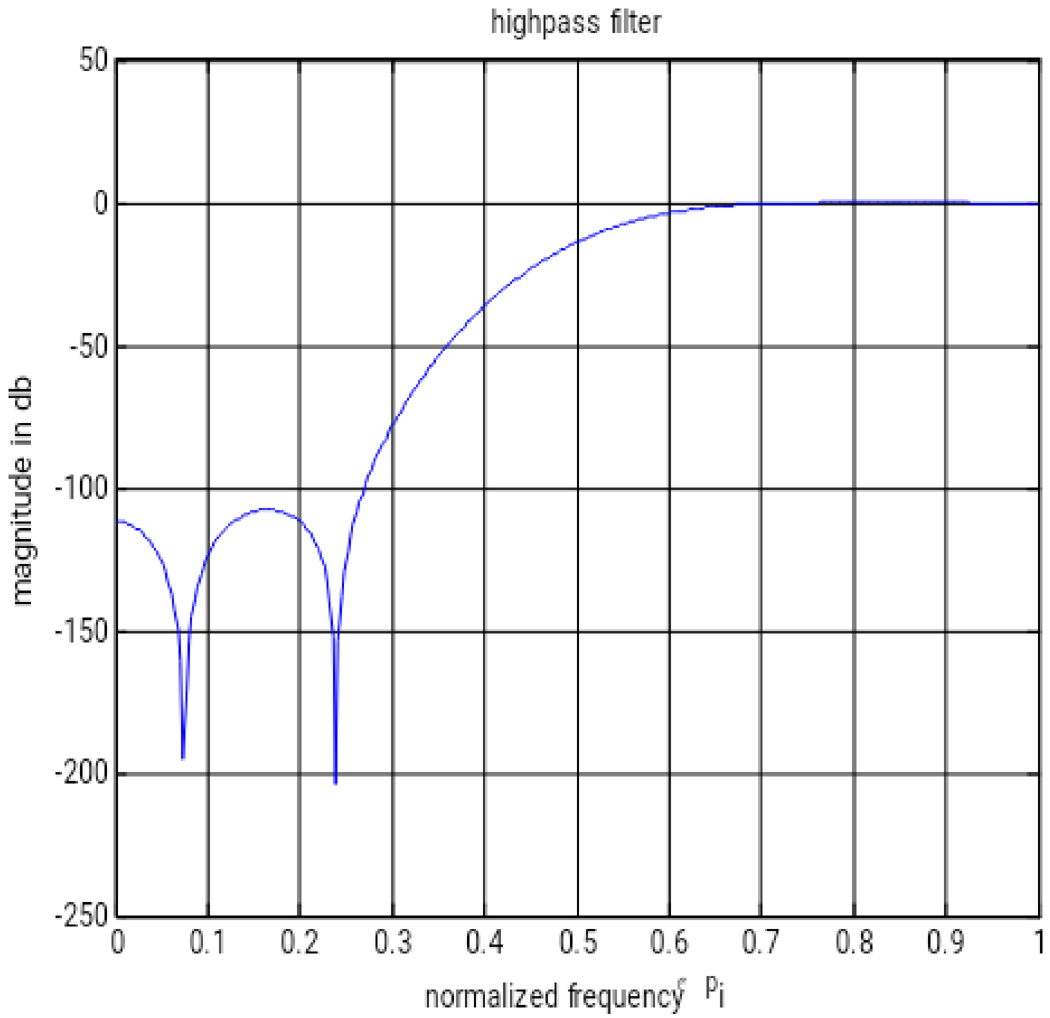

Start
Enter the N and ω_c
Calculate the hd(n)
Calculate h(n)using hd(n) and window function w(n)
Compute H(z)and plot the o/p
Stop

PROGRAM:

```
clc;
clear all;
close all;
N=input('enter the order of filter');
wc=(pi/2);
a=(N-1)/2;
for n=1:N
    if((n-1)==a)
        hd(n)=(pi-wc)/pi;
    else
        hd(n)=sin(pi*(n-1-a))-sin(wc*(n-1-a))/(pi*(n-1-a));
    end;
    w(n)=0.54-(0.46*cos(2*pi*(n-1)/(N-1)));
end;
h=w.*hd;
a=0:0.01:pi;
b=freqz(h,1,a);
mag=20*log(abs(b));
plot(a/pi,mag);
grid;
xlabel('normalized frequency\omega^pi');
ylabel('magnitude in db');
title('high pass filter');
```

OUTPUT:

Enter the order of filter 15



RESULT:

Thus the FIR high Pass Filter using hamming window was designed and executed using MATLAB.

Ex. no: 5.C	**FIR BAND PASS FILTER USING HANNING WINDOW**
Date :	

AIM:

To design a FIR band pass filter using hanning window and to plot the frequency response.

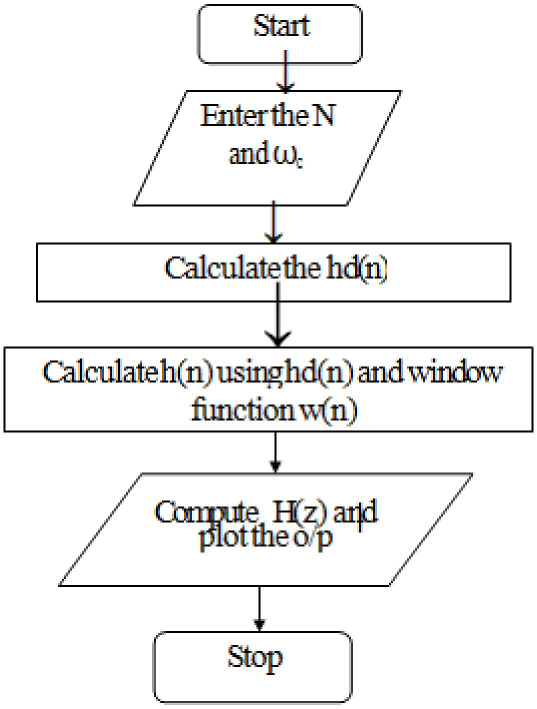
SOFTWARE REQUIRED:

MATLAB (R2020) Software.

ALGORITHM:

- Start the program
- Get the cut-off frequency & order of the filter
- For the given specifications, find the desired impulse response hd(n)
- Find the corresponding window coefficients w(n) for the given order of the filter
- Find the filter coefficients & transfer function for the given specifications
- Obtain the magnitude response & phase response
- Plot the magnitude response and phase response
- Stop the program

FLOWCHART:

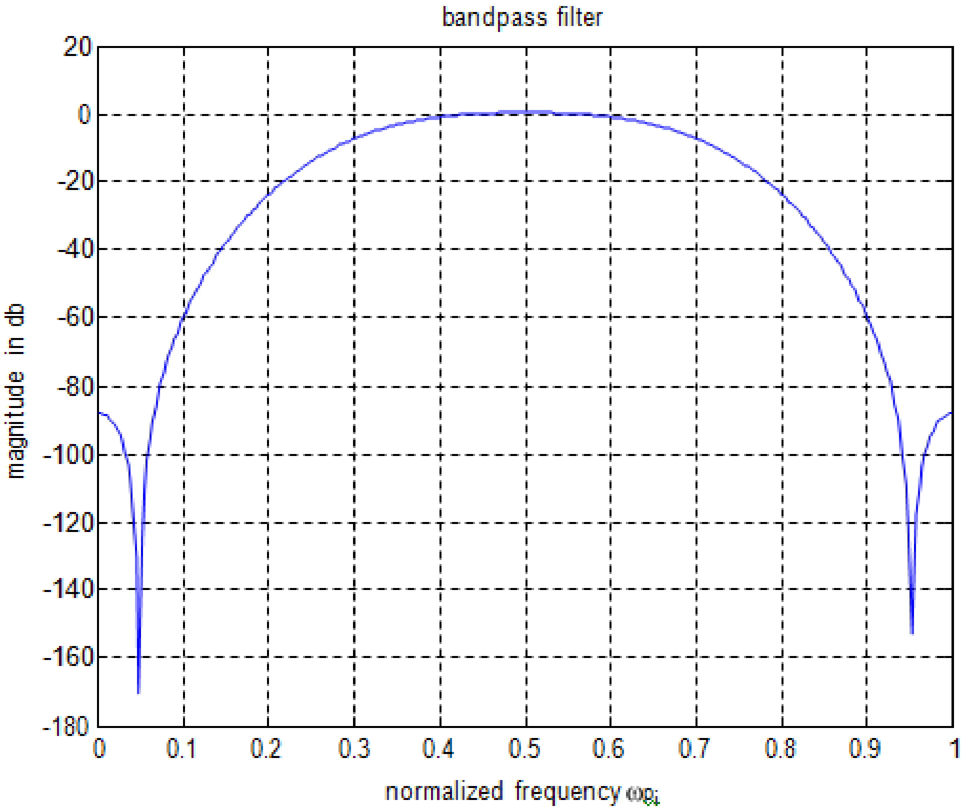

Start
Enter the N and ω_c
Calculate the hd(n)
Calculate h(n) using hd(n) and window function w(n)
Compute H(z) and plot the o/p
Stop

PROGRAM:

```
clc;
clear all;
close all;
N=input('enter the order of filter');
wc1=(pi/4);
wc2=(3*pi)/4;
a=(N-1)/2;
for n=1:N
    if((n-1)==a)
        hd(n)=(wc2-wc1)/pi;
    else
       hd(n)=(sin(wc2*(n-1-a))-sin(wc1*(n-1-a)))/(pi*(n-1-a));
    end;
    w(n)=0.5+(0.5*cos(2*pi*(n-1-a)/(N-1)));
end;
h=w.*hd;
a=0:0.01:pi;
b=freqz(h,1,a);
mag=20*log(abs(b));
plot(a/pi,mag);
grid;
xlabel('normalized frequency\omega^pi');
ylabel('magnitude in db');
title('bandpass filter');
```

OUTPUT:

```
Enter the order of filter  17
```



RESULT:

Thus the FIR BPF using Hanning window was designed and executed using MATLAB.

Ex. no: 5.D	**FIR BANDSTOP FILTER USING HANNING WINDOW**
Date:	

AIM:

To design a FIR band stop filter using hanning window and to plot the frequency response.

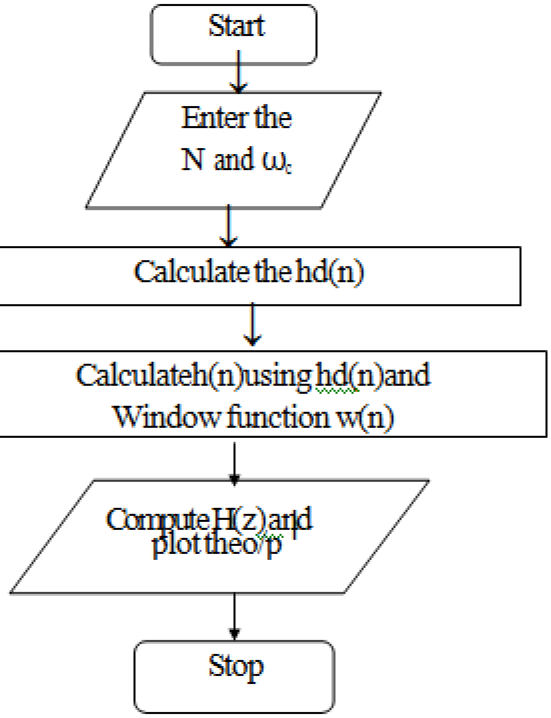
SOFTWARE REQUIRED:

MATLAB (R2020) Software.

ALGORITHM:

- Start the program
- Get the cut-off frequency& order of the filter
- For the given specifications, find the desired impulse response hd(n)
- Find the corresponding window coefficients w(n) for the given order of the filter
- Find the filter coefficients & transfer function for the given specifications
- Obtain the magnitude response & phase response
- Plot the magnitude response and phase response
- Stop the program

FLOW CHART:

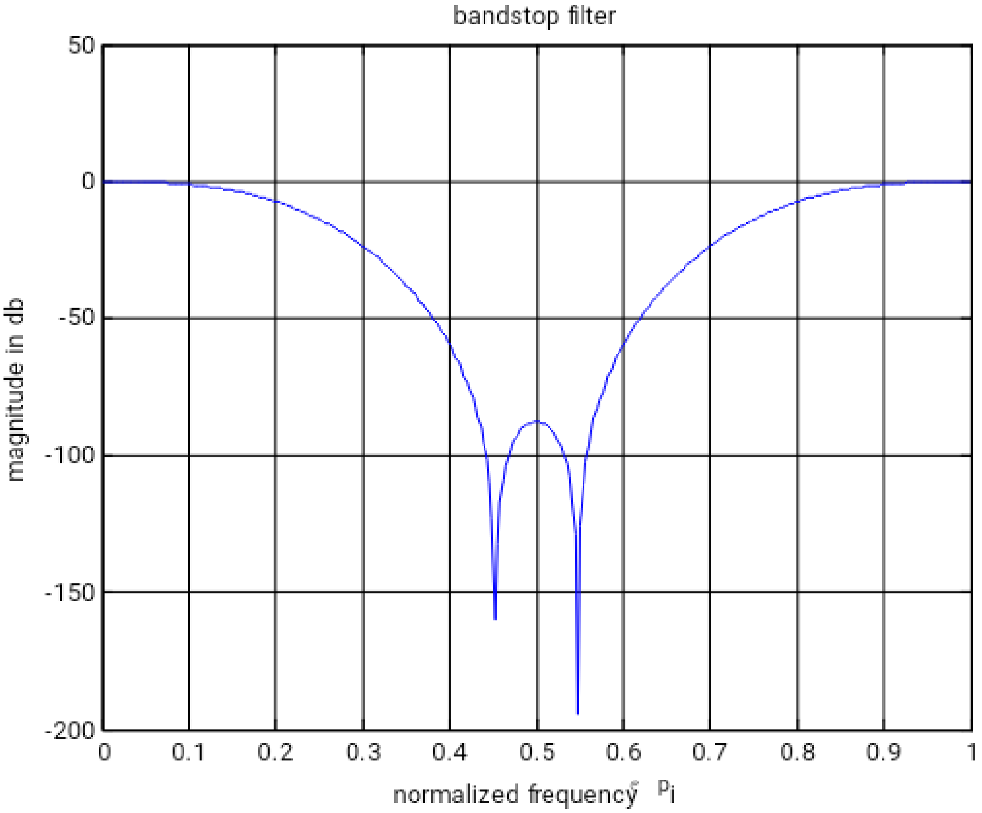


PROGRAM:

```
clc;
clear all;
close all;
N=input('enter the order of filter');
wc1=(pi/4);
wc2=(3*pi)/4;
a=(N-1)/2;
for n=1:N
   if((n-1)==a)
      hd(n)=1-(wc1+pi-wc2)/pi;
   else
     hd(n)=(sin(wc1*(n-1-a))-sin(wc2*(n-1-a)))/(pi*(n-1-a));
   end;
   w(n)=0.5+(0.5*cos(2*pi*(n-1-a)/(N-1)));
end;
h=w.*hd;
a=0:0.01:pi;
b=freqz(h,1,a);
mag=20*log(abs(b));
plot(a/pi,mag);
grid;
xlabel('normalized frequency\omega^pi');
ylabel('magnitude in db');
title('bandstop filter');
```

OUTPUT:

Enter the order of filter 17



RESULT:

Thus the FIR-Band Stop Filter using Hanning window was designed and executed using MATLAB.

Ex. no: 6. A	IIR BUTTERWORTH LOW PASS FILTER
Date:	

AIM:

To design an IIR butterworth low pass filter and to plot magnitude & phase response.

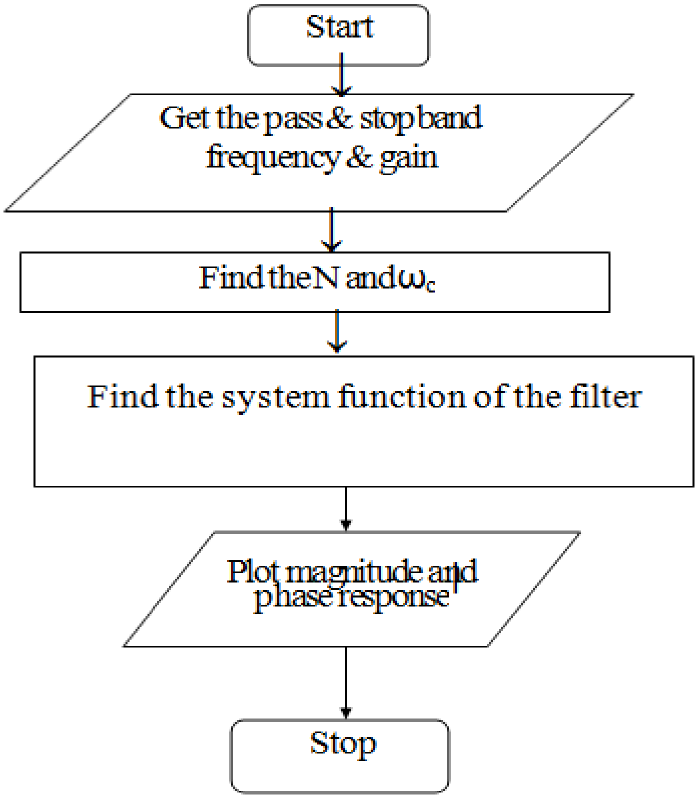
SOFTWARE REQUIRED:

MATLAB (R2020) Software.

ALGORITHM:

- Start the program
- Get the pass band attenuation(a_p) & stop band attenuation (a_s)
- Get the pass band frequency (f_p) & stop band frequency (f_s) & sampling frequency
- Find the order of the Butterworth Lowpass filter
- Find the filter coefficients & transfer function for the given specifications
- Obtain the magnitude response & phase response
- Plot the magnitude response and phase response
- Stop the program

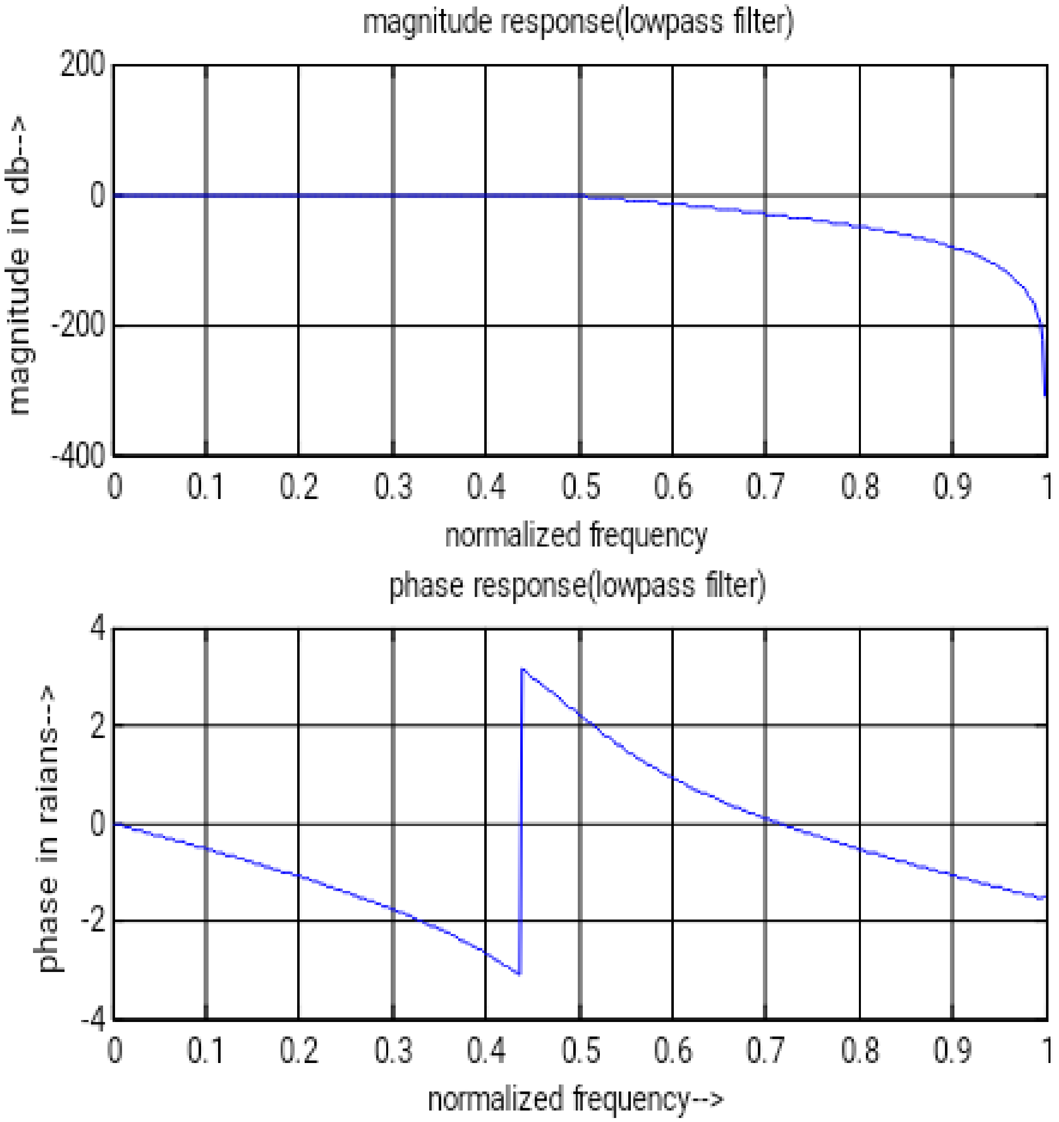
FLOWCHART:


Start
Get the pass & stop band frequency & gain
Find the N and ωc
Find the system function of the filter
Plot magnitude and phase response
Stop

PROGRAM:

```
clc;
clear all;
close all;
ap=0.5;
as=50;
fp=1000;
fs=2000;
f=5000;
wp=2*fp/f;
ws=2*fs/f;
wn=[wp,ws];

%TO FIND THE CUT OFF FREQ & ORDER OF FILTER

[n,wn]=buttord(wp,ws,ap,as);
[b,a]=butter(n,wn,'low');
w=0:0.01:pi;
h=freqz(b,a,w);
p=angle(h);
mag=20*log10(abs(h));
subplot(2,1,1);
plot(w/pi,mag);
grid;
xlabel('normalized frequency');
ylabel('magnitude in db-->');
title('magnitude response(low pass filter)');
subplot(2,1,2);
plot(w/pi,p);
grid;
xlabel('normalized frequency-->');
ylabel('phase in radians-->');
title('phase response(low pass filter)');
```

OUTPUT:



RESULT:

Thus the IIR butterworth low pass filter was designed and executed using MATLAB.

Ex. no: 6.B	**IIR BUTTERWORTH HIGH PASS FILTER**
Date:	

AIM:

To design an IIR Butterworth high pass filter and to plot magnitude & phase response.

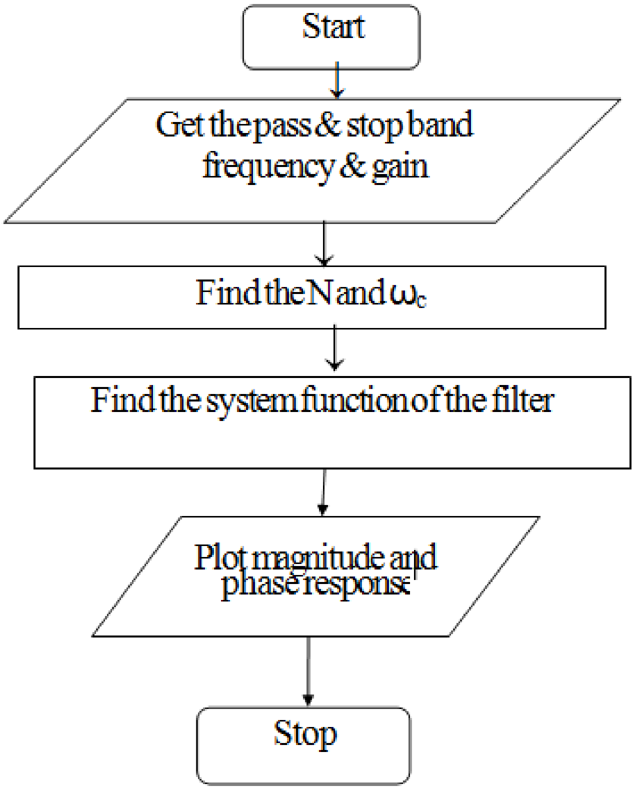
SOFTWARE REQUIRED:

MATLAB (R2020) Software.

ALGORITHM:

- Start the program
- Get the pass band attenuation(a_p) & stop band attenuation (a_s)
- Get the pass band frequency (f_p) & stop band frequency (f_s) & sampling frequency
- Find the order of the Butterworth highpass filter
- Find the filter coefficients & transfer function for the given specifications
- Obtain the magnitude response & phase response
- Plot the magnitude response and phase response
- Stop the program

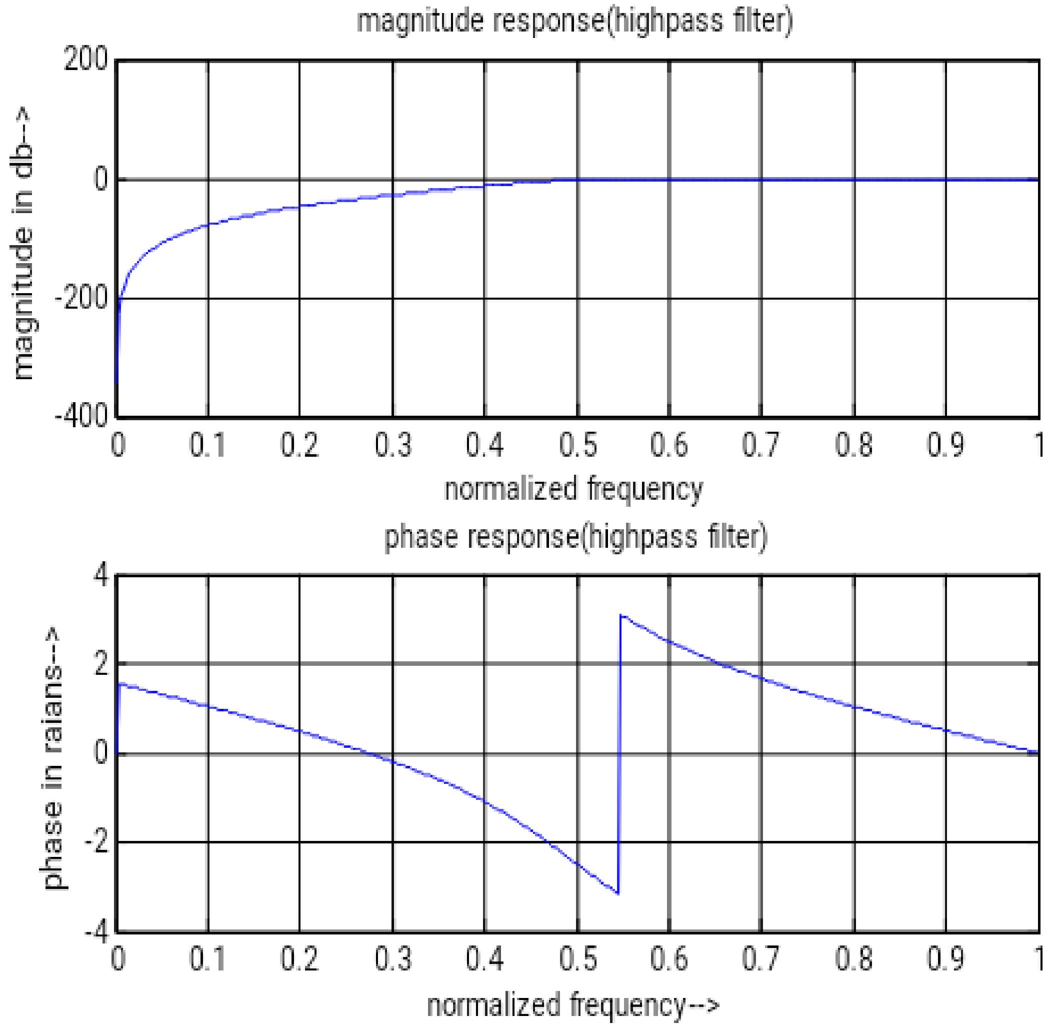
FLOWCHART:


Start
Get the pass & stop band frequency & gain
Find the N and ωc
Find the system function of the filter
Plot magnitude and phase response
Stop

PROGRAM:

```
clc;
clear all;
close all;
ap=0.5;
as=50;
fp=1000;
fs=2000;
f=5000;
wp=2*fp/f;
ws=2*fs/f;
wn=[wp,ws];

%TO FIND THE CUT OFF FREQ&ORDER OF FILTER
[n,wn]=buttord(wp,ws,ap,as);
[b,a]=butter(n,wn,'high');
w=0:0.01:pi;
h=freqz(b,a,w);
p=angle(h);
mag=20*log10(abs(h));
subplot(2,1,1);
plot(w/pi,mag);
grid;
xlabel('normalized frequency');
ylabel('magnitude in db-->');
title('magnitude response(highpass filter)');
subplot(2,1,2);
plot(w/pi,p);
grid;
xlabel('normalized frequency-->');
ylabel('phase in radians-->');
title('phase response(highpass filter)');
```

OUTPUT:



RESULT:

Thus the IIR Butterworth High Pass Filter was designed and executed using MATLAB.

Ex. no: 6. C	**IIR BUTTERWORTH BAND PASS FILTER**
Date :	

AIM:

To design an IIR Butterworth Band pass filter and to plot magnitude & phase response.

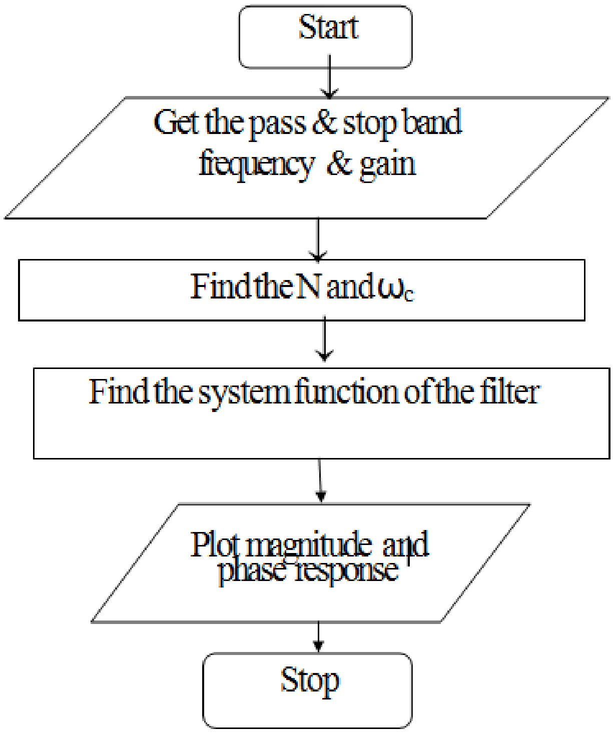
SOFTWARE REQUIRED:

MATLAB (R2020) Software.

ALGORITHM:

- Start the program
- Get the pass band attenuation(a_p) & stop band attenuation (a_s)
- Get the pass band frequency (f_p) & stop band frequency (f_s) & sampling frequency
- Find the order of the Butterworth bandpass filter
- Find the filter coefficients & transfer function for the given specifications
- Obtain the magnitude response & phase response
- Plot the magnitude response and phase response
- Stop the program

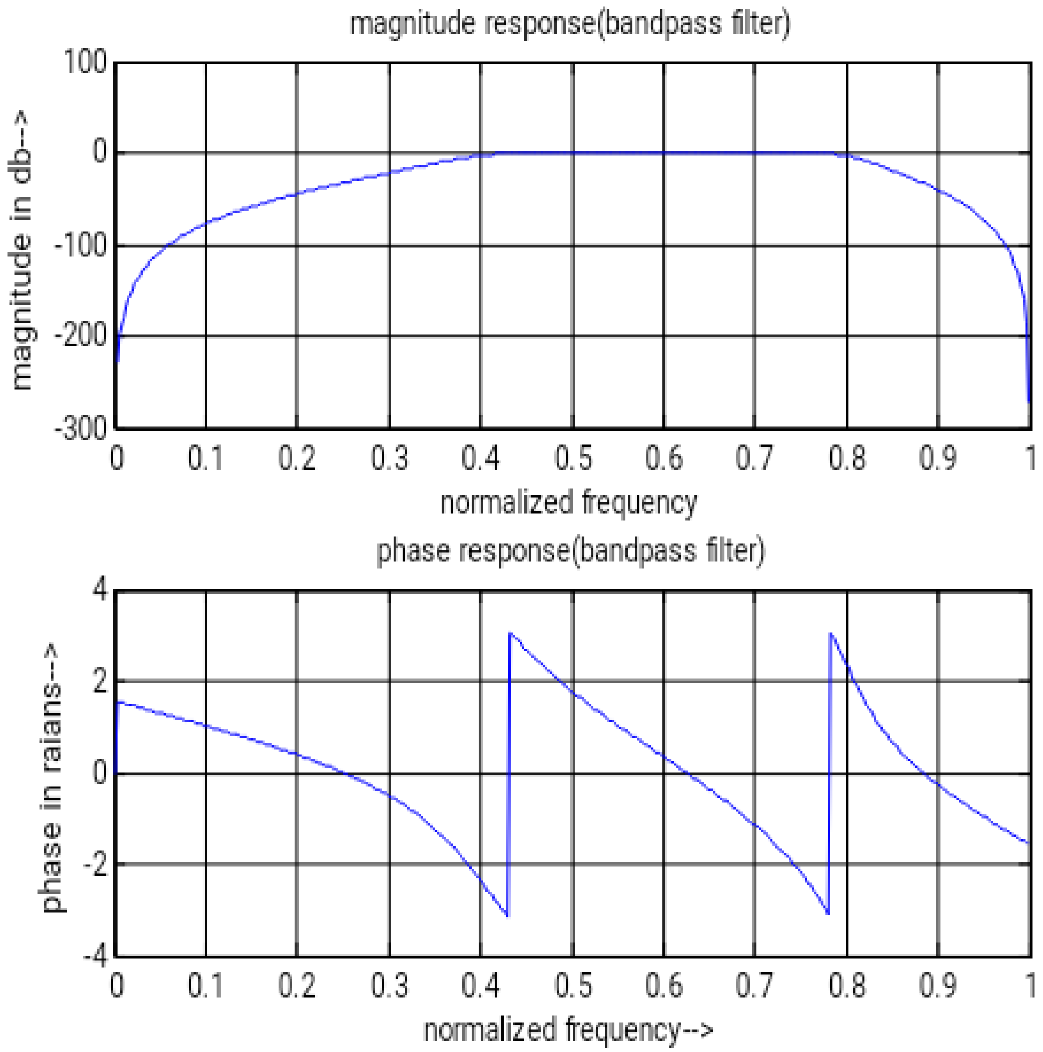
FLOWCHART:


Start
Get the pass & stop band frequency & gain
Find the N and ωc
Find the system function of the filter
Plot magnitude and phase response
Stop

PROGRAM:

```
clc;
clear all;
close all;
ap=0.5;
as=50;
fp=1000;
fs=2000;
f=5000;
wp=2*fp/f;
ws=2*fs/f;
wn=[wp,ws];
[n]=buttord(wp,ws,ap,as);
[b,a]=butter(n,wn,'bandpass');
w=0:0.01:pi;
h=freqz(b,a,w);
p=angle(h);
mag=20*log10(abs(h));
subplot(2,1,1);
plot(w/pi,mag);
grid;
xlabel('normalized frequency');
ylabel('magnitude in db-->');
title('magnitude response(band pass filter)');
subplot(2,1,2);
plot(w/pi,p);
grid;
xlabel('normalized frequency-->');
ylabel('phase in radians-->');
title('phase response(band pass filter)');
```

OUTPUT:



RESULT:

Thus the IIR Butterworth band pass filter was designed and executed using MATLAB.

Ex. no: 6.D	**IIR BUTTERWORTH BAND STOP FILTER**
Date :	

AIM:

To design an IIR butterworth band stop filter and to plot magnitude & phase response.

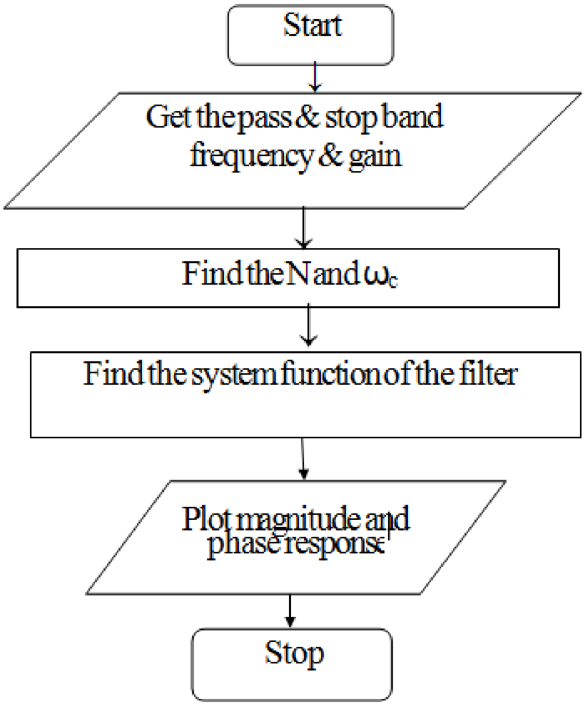
SOFTWARE REQUIRED:

MATLAB (R2020) Software.

ALGORITHM:

- Start the program
- Get the pass band attenuation(a_p) & stop band attenuation (a_s)
- Get the pass band frequency (f_p) & stop band frequency (f_s) & sampling frequency
- Find the order of the Butterworth bandstop filter
- Find the filter coefficients & transfer function for the given specifications
- Obtain the magnitude response & phase response
- Plot the magnitude response and phase response
- Stop the program

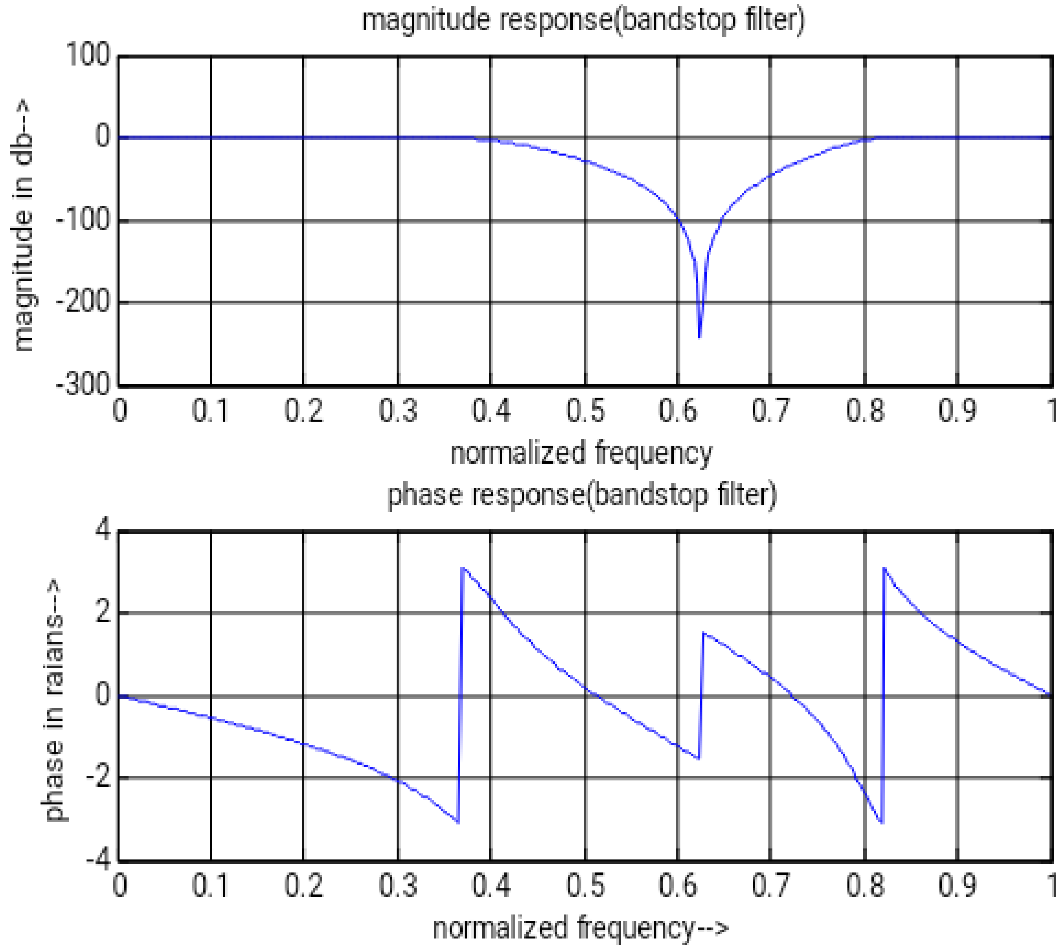
FLOWCHART:


Start
Get the pass & stop band frequency & gain
Find the N and ωc
Find the system function of the filter
Plot magnitude and phase response
Stop

PROGRAM:

```
clc;
clear all;
close all;
ap=0.5;
as=50;
fp=1000;
fs=2000;
f=5000;
wp=2*fp/f;
ws=2*fs/f;
wn=[wp,ws];
[n]=buttord(wp,ws,ap,as);
[b,a]=butter(n,wn,'stop');
w=0:0.01:pi;
h=freqz(b,a,w);
p=angle(h);
mag=20*log10(abs(h));
subplot(2,1,1);
plot(w/pi,mag);
grid;
xlabel('normalized frequency');
ylabel('magnitude in db-->');
title('magnitude response(band stop filter)');
subplot(2,1,2);
plot(w/pi,p);
grid;
xlabel('normalized frequency-->');
ylabel('phase in radians-->');
title('phase response(bandstop filter)');
```

OUTPUT:



RESULT:

Thus the IIR Butterworth Band Stop Filter was designed and executed using MATLAB.

Ex. no: 6. E	**IMPLEMENTATION OF CHEBYSHEV TYPE-1 ANALOG FILTERS**
Date:	

AIM:

To design and execute Chebyshev Type-I analog Low pass filter using MATLAB.

SOFTWARE REQUIRED:

MATLAB (R2020) Software.

ALGORITHM

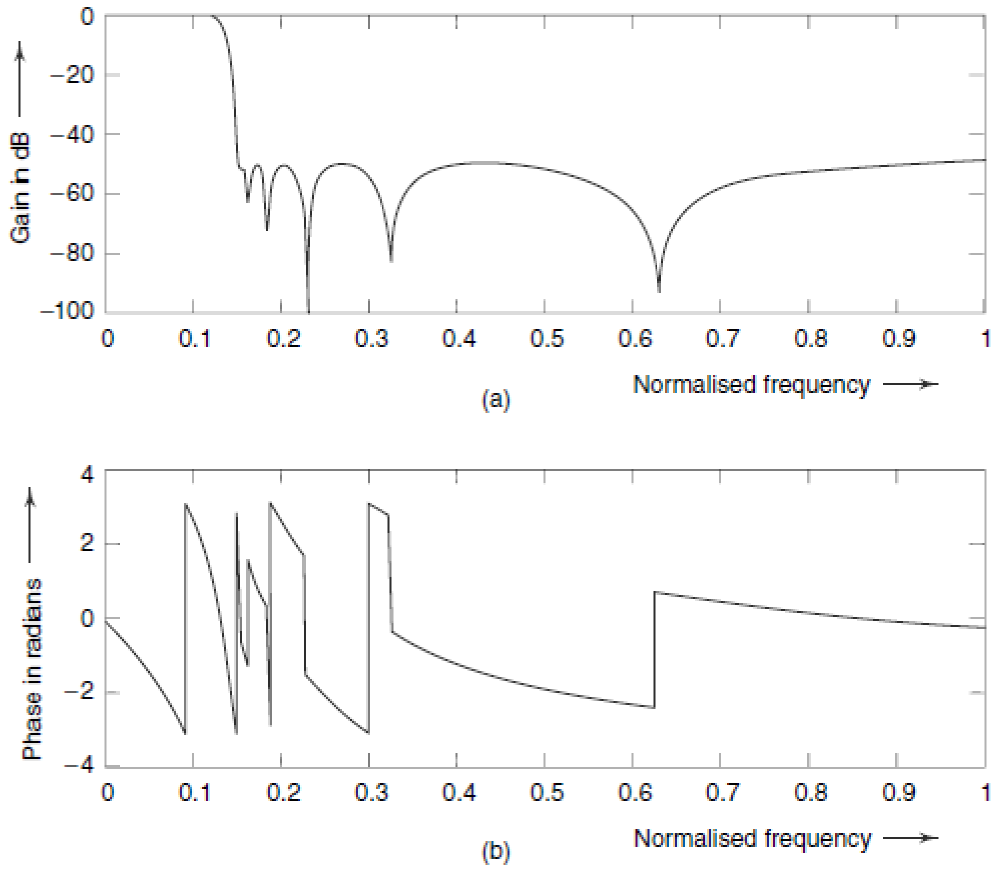
- Get the passband and stopband ripples
- Get the passband and stopband edge frequencies
- Get the sampling frequency
- Calculate the order of the filter.
- Find the filter coefficients
- Draw the magnitude and phase responses.

PROGRAM:

```
clc;
close all;clear all;
format long
rp=input('enter the passband ripple...');
rs=input('enter the stopband ripple...');
wp=input('enter the passband freq...');
ws=input('enter the stopband freq...');
fs=input('enter the sampling freq...');
w1=2*wp/fs;w2=2*ws/fs;
[n,wn]=cheb1ord(w1,w2,rp,rs,'s');
[b,a]=cheby1(n,rp,wn,'s');
W=0:.01:pi;
[h,om]=freqs(b,a,W);
M=20*log10(abs(h));
An=angle(h);
subplot(2,1,1);
plot(om/pi,M);
ylabel('Gain in dB --.');
xlabel('(a) Normalised frequency --.');
subplot(2,1,2);
plot(om/pi,An);
xlabel('b) Normalised frequency --.');
ylabel('Phase in radians --.');
```

OUTPUT:

```
enter the passband ripple... 0.23
enter the stopband ripple... 47
enter the passband freq... 1300
enter the stopband freq... 1550
enter the sampling freq... 7800
```



(a)

(b)

RESULT:

Thus the Chebyshev Type-I analog Low pass filter was designed and executed using MATLAB.

Ex. no: 6. F	**IMPLEMENTATION OF CHEBYSHEV TYPE-2 ANALOG FILTERS**
Date:	

AIM:

To design and execute Chebyshev Type-II analog High pass filter using MATLAB.

SOFTWARE REQUIRED:

MATLAB (R2020) Software.

ALGORITHM

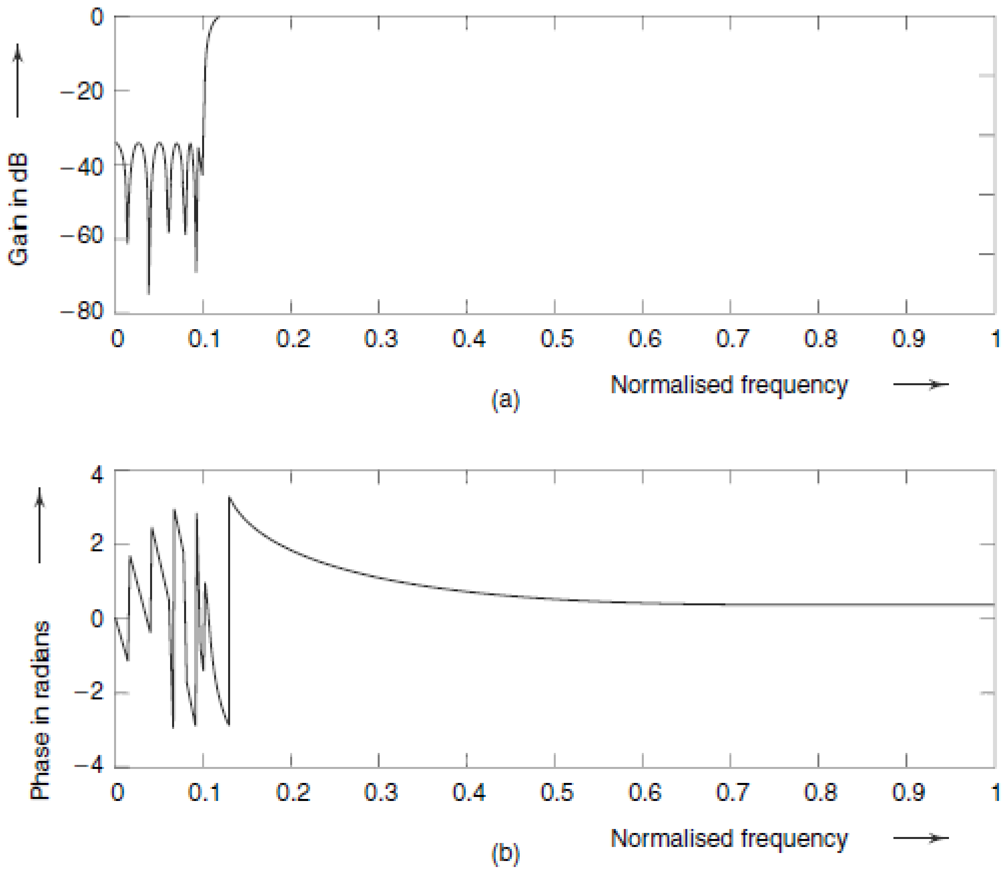
- Get the passband and stopband ripples
- Get the passband and stopband edge frequencies
- Get the sampling frequency
- Calculate the order of the filter.
- Find the filter coefficients
- Draw the magnitude and phase responses.

PROGRAM:

```
clc;
close all;clear all;
format long
rp=input('enter the passband ripple...');
rs=input('enter the stopband ripple...');
wp=input('enter the passband freq...');
ws=input('enter the stopband freq...');
fs=input('enter the sampling freq...');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=cheb2ord(w1,w2,rp,rs,'s');
[b,a]=cheby2(n,rs,wn,'high','s');
w=0:.01:pi;
[h,om]=freqs(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1);
plot(om/pi,m);
ylabel('Gain in dB --.');
xlabel('(a) Normalised frequency --.');
subplot(2,1,2);
plot(om/pi,an);
xlabel('(b) Normalised frequency --.');
ylabel('Phase in radians --.');
```

OUTPUT:

```
enter the passband ripple... 0.34
enter the stopband ripple... 34
enter the passband freq... 1400
enter the stopband freq... 1600
enter the sampling freq... 10000
```



Chebyshev Type - 2 High-pass Analog Filter
(a) Amplitude Response and (b) Phase Response

RESULT:

Thus the Chebyshev Type-II analog HIGH pass filter was designed and executed using MATLAB.

Ex. no:7(a)	**DESIGN OF DECIMATION PROCESS IN MULTIRATE FILTERS**
Date:	

AIM:

To perform the down sampling process for the given sequence.

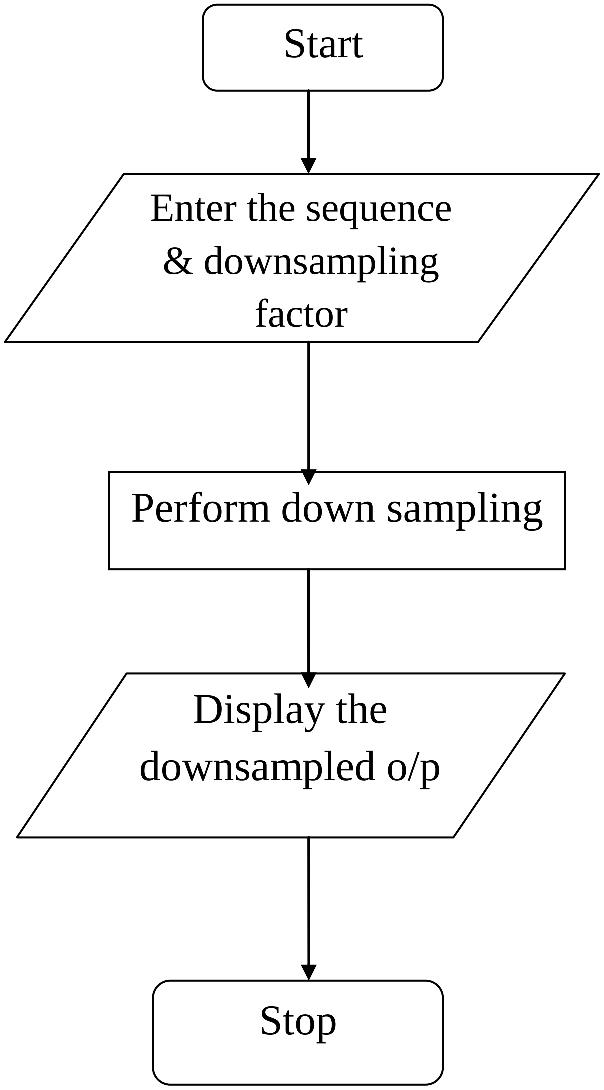
SOFTWARE REQUIRED:

MATLAB (R2020) software

ALGORITHM:

- Start the program
- Generate the input signal
- Enter the down sampling factor
- Sample the given signal to get sampled signal
- Perform down sampling by neglecting the samples at specified intervals
- Plot the input signal, sampled signal & down sampled signal
- Stop the program

FLOW CHART:


Start
Enter the sequence & downsampling factor
Perform down sampling
Display the downsampled o/p
Stop

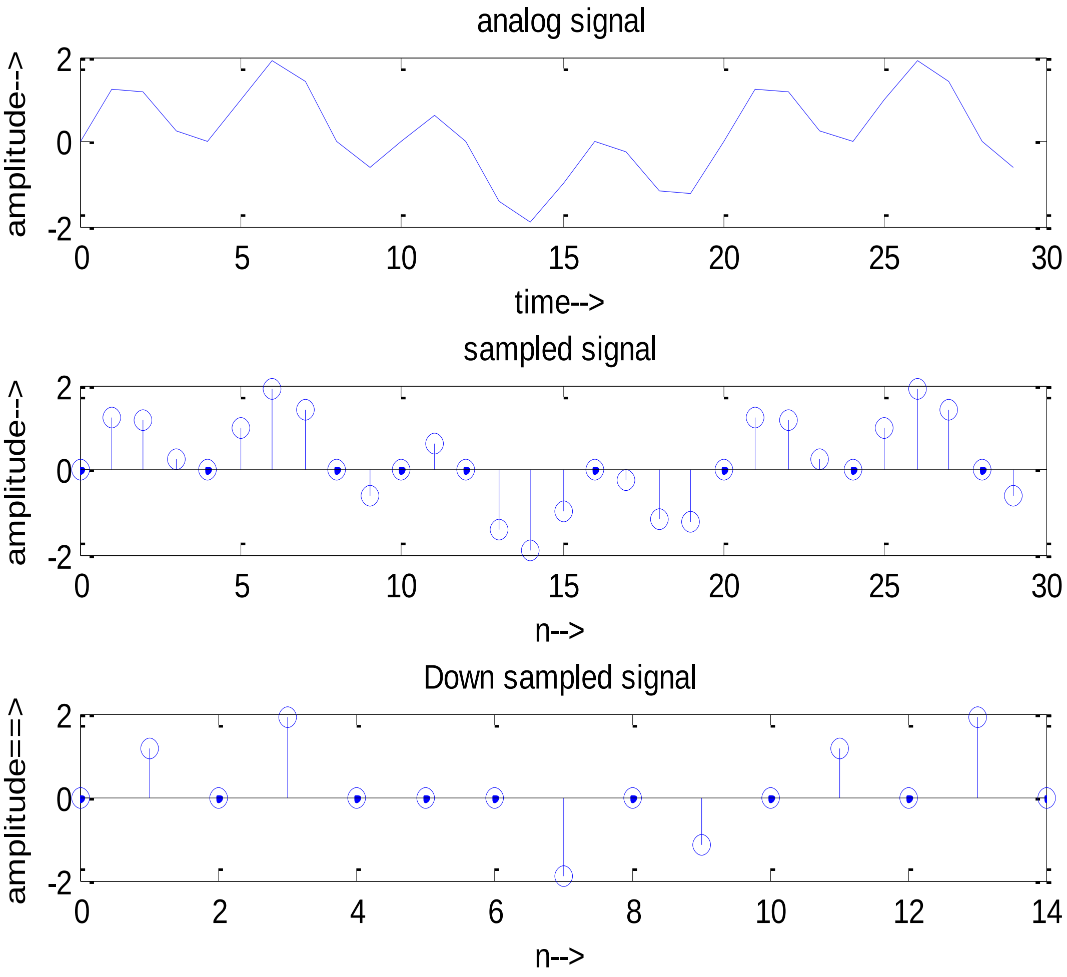
DOWN SAMPLING

PROGRAM:

```
clc;
clear all;
close all;
N=input('Enter the sequence length N:');
M=input('Enter the down sampling factor M:');
f1=0.05;
f2=0.2;
t=0:1:N-1;
x=sin(2*pi*f1*t)+sin(2*pi*f2*t);
x1=x(1:M:N);
t1=1:1:N/M;
subplot(3,1,1);
plot(t,x);
xlabel('time-->');
ylabel('amplitude-->');
title('Input analog signal');
subplot(3,1,2);
stem(t,x);
xlabel('n-->');
ylabel('amplitude-->');
title('Sampled signal');
subplot(3,1,3);
stem(t1-1,x1);
xlabel('n-->');
ylabel('amplitude==>');
title('Down sampled signal');
```

OUTPUT:

```
Enter the sequence length N:30
Enter the down sampling factor M:2
```



RESULT:

Thus the down sampling of the given signal was performed and executed using MATLAB

Ex. no:7(b)	**DESIGN OF INTERPOLATON PROCESS IN MULTIRATE FILTERS**
Date:	

AIM:

To perform the upsampling process for the given sequence.

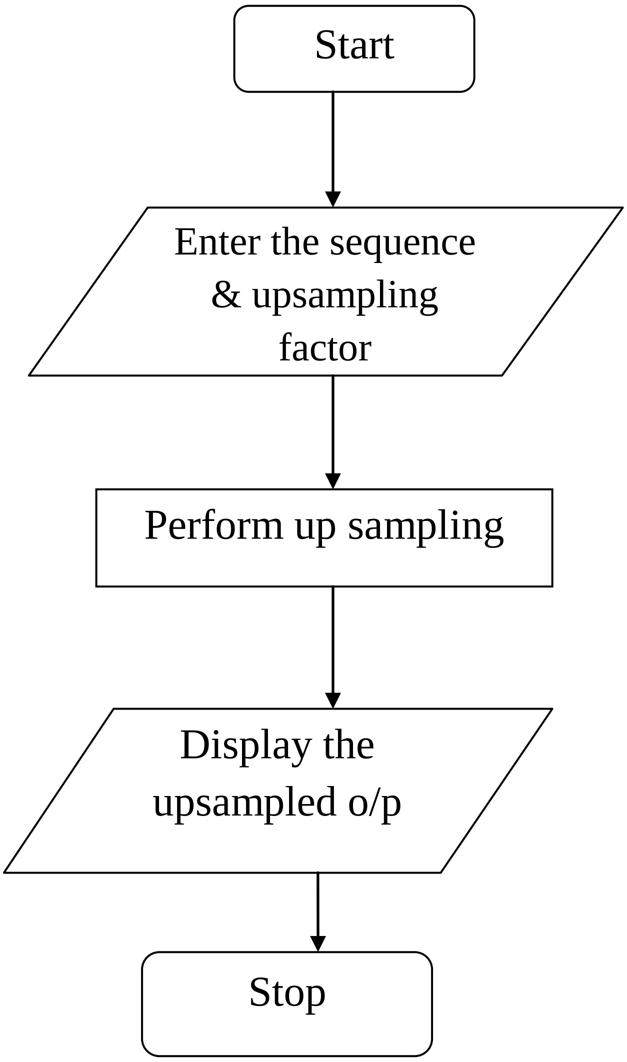
SOFTWARE REQUIRED:

MATLAB (R2020) software

ALGORITHM:

- Start the program
- Generate the input signal
- Enter the up sampling factor
- Sample the given signal to get sampled signal
- Perform up sampling by introducing zero’s in between the samples
- Plot the input signal, sampled signal & up sampled signal
- Stop the program

FLOW CHART:



Start
Enter the sequence & upsampling factor
Perform up sampling
Display the upsampled o/p
Stop

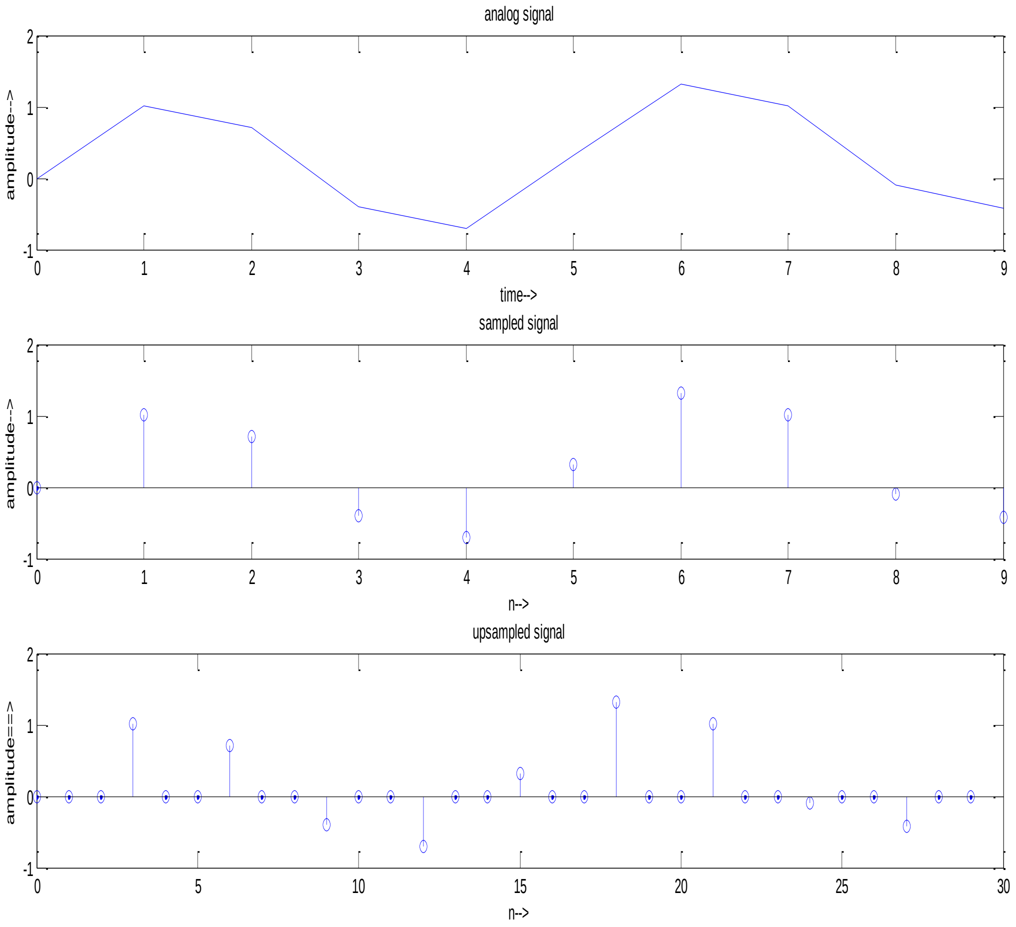
UPSAMPLING

PROGRAM:

```
clc;
clear all;
close all;
N=input('Enter the sequence length N:');
L=input('Enter the upsampling factor L:');
f1=0.01;
f2=0.2;
t=0:1:N-1;
x=sin(2*pi*f1*t)+sin(2*pi*f2*t);
x1=[zeros(1,L*N)];
t1=1:1:L*N;
j=1:L:L*N;
x1(j)=x;
subplot(3,1,1);
plot(t,x);
xlabel('time-->');
ylabel('amplitude-->');
title('Input analog signal');
subplot(3,1,2);
stem(t,x);
xlabel('n-->');
ylabel('amplitude-->');
title('sampled signal');
subplot(3,1,3);
stem(t1-1,x1);
xlabel('n-->');
ylabel('amplitude==>');
title('upsampled signal');
```

OUTPUT:

```
Enter the sequence length N:10
Enter the upsampling factor L:3
```



RESULT:

Thus the upsampling of the given signal was performed and executed using MATLAB

Ex. no: 7(c)	**DESIGN OF SAMPLING RATE CONVERTOR BY RATIONAL FACTOR**
Date:	

AIM:

To perform sampling rate conversion by rational factor I/D'

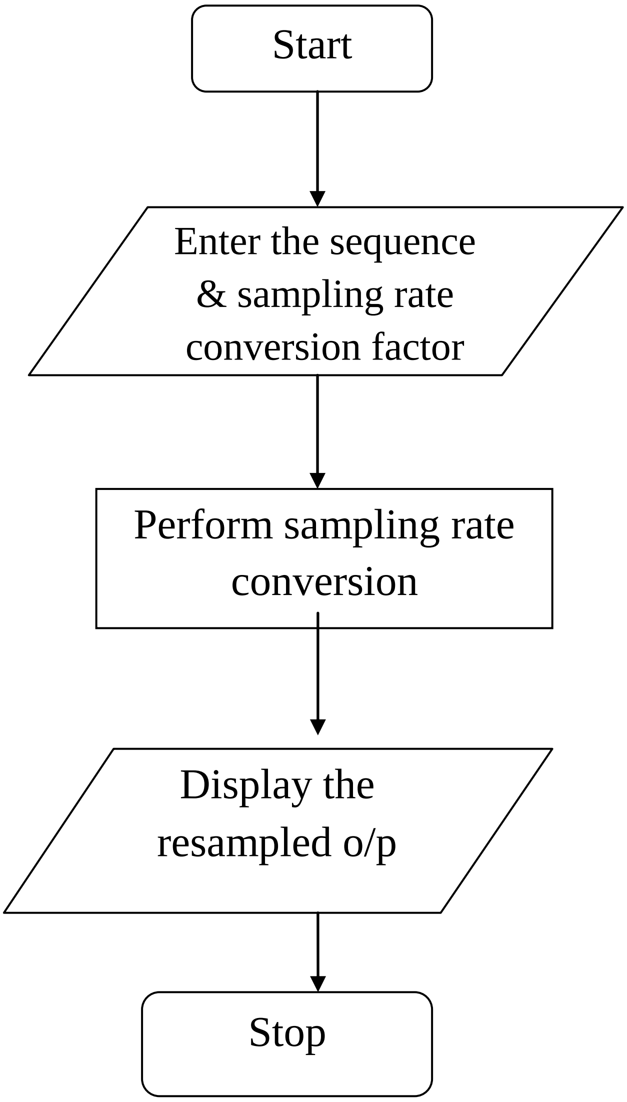
SOFTWARE REQUIRED:

MATLAB (R2020) version.

ALGORITHM:

- Start the program
- Generate the input signal
- Sample the given signal to get sampled signal
- Perform sampling rate conversion by rational factor I/D
- Plot the input signal, resampled output for factor I/D
- Stop the program

FLOW CHART:

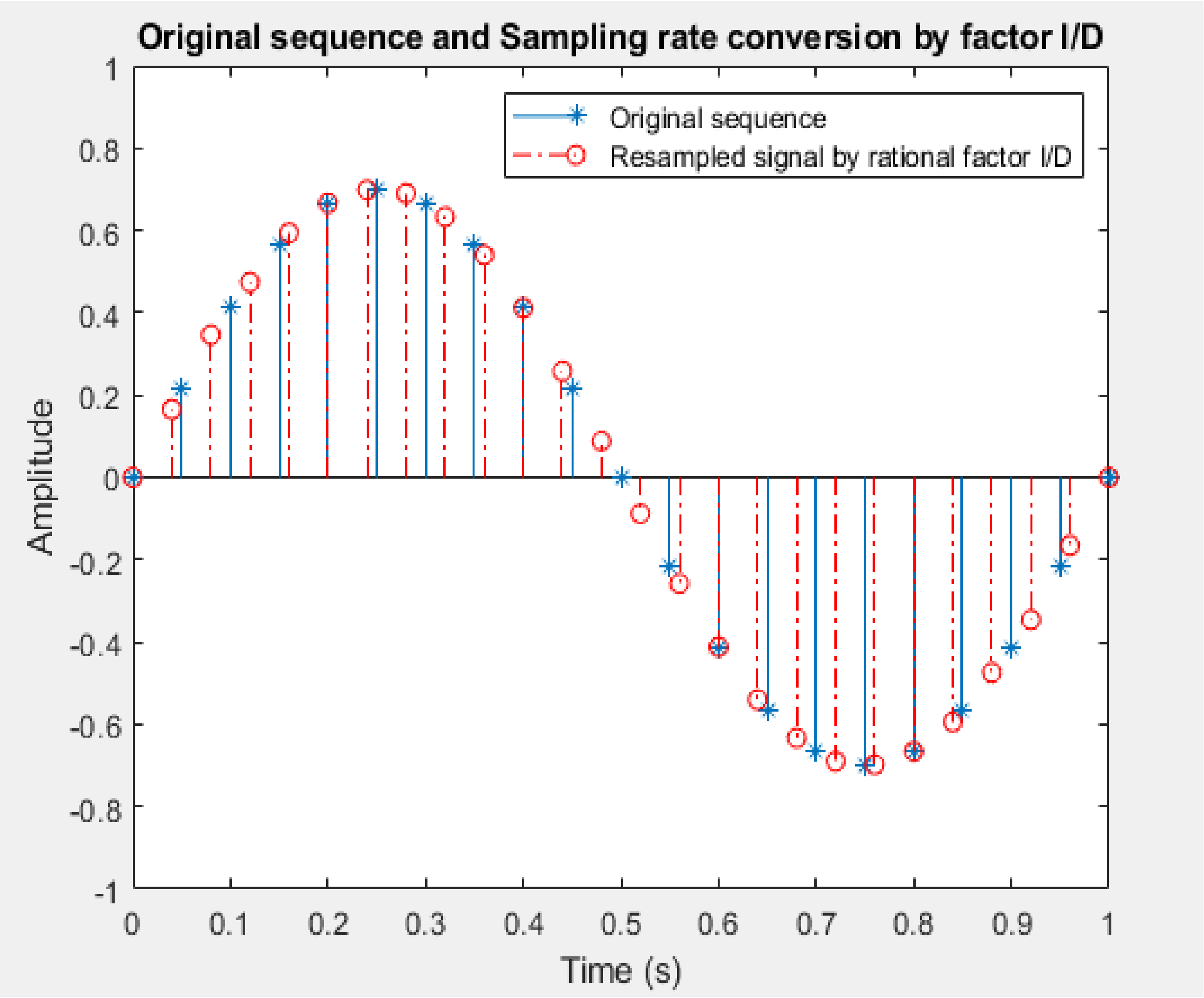

Start
Enter the sequence & sampling rate conversion factor
Perform sampling rate conversion
Display the resampled o/p
Stop

SAMPLING RATE CONVERTOR BY RATIONAL FACTOR I/D

PROGRAM:

```
clc;
clear all;
close all;    %Sampling rate conversion by factor I/D = 5/4
Fx=20;      %Original sampling frequency in Hz
Tx=1/Fx;      %Original sampling period in seconds
tx=0:Tx:1; % Time vector tx
x=0.7*sin(2*pi*tx); % Original sequence
y=resample(x,5,4);  % Re-sampling by rational factor I/D
ty=(0:(length(y)-1))*4*Tx/5;  % New time vector ty
figure(1)
stem(tx,x,'*')
hold on
stem(ty,y,'-.r')
title('Original sequence and Sampling rate conversion by
factor I/D')
legend('Original sequence','Resampled signal by rational
factor I/D')
xlabel('Time (s)'),
ylabel('Amplitude'),
axis([0,1,-1,1])
```

OUTPUT:



RESULT:

Thus the sampling rate conversion for rational factor I/D was performed using MATLAB.

Ex. no: 8	**STUDY OF ARCHITECTURE OF DIGITAL SIGNAL PROCESSOR**
Date:	

AIM:

To study and understand the architecture of DSP processor.

ARCHITECTURE:

Texas Instruments TMS320 is a blanket name for a series of digital signal processors (DSPs) from Texas Instruments. It was introduced on April 8, 1983 through the TMS32010 processor, which was then the fastest DSP on the market.

The processor is available in many different variants, some with fixed-point arithmetic and some with floating point arithmetic. The floating point DSP TMS320C3x, which exploits delayed branch logic, has as many as three delay slots.

The flexibility of this line of processors has led to it being used not merely as a co-processor for digital signal processing but also as a main CPU. Newer implementations support standard IEEE JTAG control for boundary scan and/or in-circuit debugging.

The original TMS32010 and its subsequent variants is an example of a CPU with a modified Harvard architecture, which features separate address spaces for instruction and data memory but the ability to read data values from instruction memory. The TMS32010 featured a fast multiply-and-accumulate useful in both DSP applications as well as transformations used in computer graphics. The graphics controller card for the Apollo Computer DN570 Workstation, released in 1985, was based on the TMS32010 and could transform 20,000 2D vectors every second.

The 'C5x uses an advanced, modified Harvard-type architecture based on the 'C25 architecture and maximizes processing power with separate buses for program memory and data memory. The instruction set supports data transfers between the two memory spaces.

The functional block diagram of TMS320C5x is shown in below figure. It can be divided into four sub-blocks they are bus structure, on-chip memory and central processing unit (CPU) and on-chip peripherals.

BUS STRUCTURE:

Separate program and data buses allow simultaneous access to program instructions and data, providing a high degree of parallelism. For example, while data is multiplied, a previous product can be loaded into, added to, or subtracted from the accumulator and, at the same time, a new address can be generated. Such parallelism supports a powerful set of arithmetic, logic, and bit-manipulation operations that can all be performed in a single machine cycle. In addition, the 'C5x includes the control mechanisms to manage interrupts, repeated operations, and function calling.

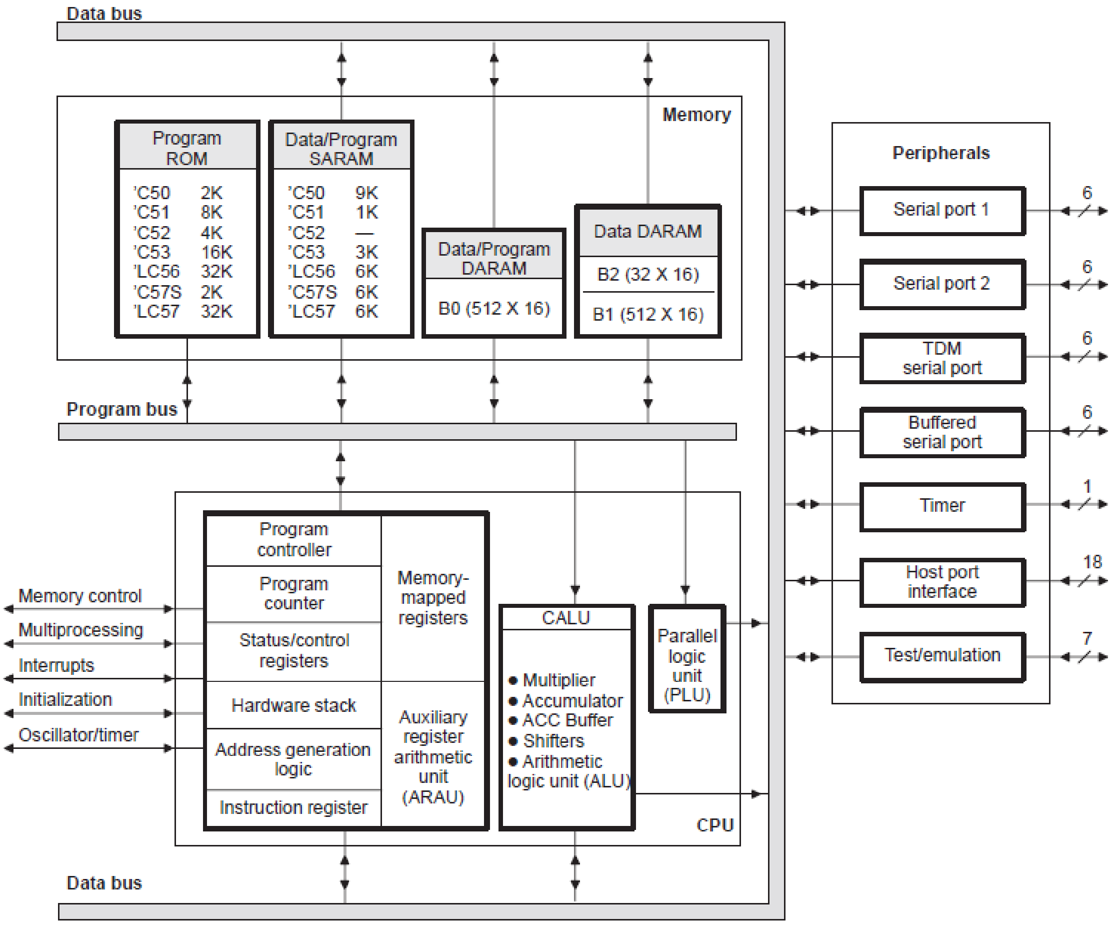


Figure 1. The functional block diagram of TMS320C50

The 'C5x architecture is built around four major buses:

i. Program bus (PB)

ii. Program address bus (PAB)

iii. Data read bus (DB)

iv. Data read address bus (DAB)

The PAB provides addresses to program memory space for both reads and writes. The PB also carries the instruction code and immediate operands from program memory space to the CPU. The DB interconnects various elements of the CPU to data memory space. The program and data buses can work together to transfer data from on-chip data memory and internal or external program memory to the multiplier for single-cycle multiply/accumulate operations.

CENTRAL PROCESSING UNIT (CPU):

The 'C5x CPU consists of these elements:

i. Central arithmetic logic unit (CALU)

ii. Parallel logic unit (PLU)

iii. Auxiliary register arithmetic unit (ARAU)

iv. Memory-mapped registers

v. Program controller

CENTRAL ARITHMETIC LOGIC UNIT (CALU) :

The CPU uses the CALU to perform 2s-complement arithmetic. The CALU consists of these elements:

i. 16-bit   16-bit multiplier

ii. 32-bit arithmetic logic unit (ALU)

iii. 32-bit accumulator (ACC)

iv. 32-bit accumulator buffer (ACCB)

v. Additional shifters at the outputs of both the accumulator and the product register (PREG)

PARALLEL LOGIC UNIT (PLU) :

The CPU includes an independent PLU, which operates separately from, but in parallel with, the ALU. The PLU performs Boolean operations or the bit manipulations required of high-speed controllers. The PLU can set, clear, test, or toggle bits in a status register, control register, or any data memory location. The PLU provides a direct logic operation path to data memory values without affecting the contents of the ACC or PREG. Results of a PLU function are writ-ten back to the original data memory location.

AUXILIARY REGISTER ARITHMETIC UNIT (ARAU) :

The CPU includes an unsigned 16-bit arithmetic logic unit that calculates indirect addresses by using inputs from the auxiliary registers (ARs), index register (INDX), and auxiliary register compare register (ARCR). The ARAU can auto index the current AR while the data memory location is being addressed and can index either by 1 or by the contents of the INDX. As a result, accessing data does not require the CALU for address manipulation; therefore, the CALU is free for other operations in parallel.

MEMORY-MAPPED REGISTERS :

The 'C5x has 96 registers mapped into page 0 of the data memory space. All 'C5x DSPs have 28 CPU registers and 16 input/output (I/O) port registers but have different numbers of peripheral and reserved registers. Since the memory-mapped registers are a component of the data memory space, they can be written to and read from in the same way as any other data memory location. The memory-mapped registers are used for indirect data address pointers, temporary storage, CPU status and control, or integer arithmetic processing through the ARAU.

PROGRAM CONTROLLER :

The program controller contains logic circuitry that decodes the operational instructions, manages the CPU pipeline, stores the status of CPU operations, and decodes the conditional operations. Parallelism of architecture lets the 'C5x perform three concurrent memory operations in any given machine cycle: fetch an instruction, read an operand, and write an operand.

The program controller consists of these elements:

i. Program counter

ii. Status and control registers

iii. Hardware stack

iv. Address generation logic

v. Instruction register

ON-CHIP MEMORY :

The 'C5x architecture contains a considerable amount of on-chip memory to aid in system performance and integration:

i. Program read-only memory (ROM)

ii. Data/program dual-access RAM (DARAM)

iii. Data/program single-access RAM (SARAM)

The 'C5x has a total address range of 224K words 16 bits. The memory space is divided into four individually selectable memory segments: 64K-word program memory space, 64K-word local data memory space, 64K-word input/ output ports, and 32K-word global data memory space.

PROGRAM ROM :

All 'C5x DSPs carry a 16-bit on-chip maskable programmable ROM. The 'C50 and 'C57S DSPs have boot loader code resident in the on-chip ROM, all other 'C5x DSPs offer the boot loader code as an option. This memory is used for booting program code from slower external ROM or EPROM to fast on-chip or external RAM. Once the custom program has been booted into RAM, the boot ROM space can be removed from pro-gram memory space by setting the MP/MC bit in the processor mode status register (PMST). The on-chip ROM is selected at reset by driving the MP/MC pin low. If the on-chip ROM is not selected, the 'C5x devices start execution from off-chip memory.
The on-chip ROM may be configured with or without boot loader code. However, the on-chip ROM is intended for your specific program.

DATA/PROGRAM DUAL-ACCESS RAM :

All 'C5x DSPs carry a 1056-word16-bit on-chip dual-access RAM (DARAM). The DARAM is divided into three individually selectable memory blocks: 512-word data or program DARAM block B0, 512-word data DARAM block B1, and 32-word data DARAM block B2. The DARAM is primarily intended to store data values but, when needed, can be used to store programs as well. DARAM blocks B1 and B2 are always configured as data memory; however, DARAM block B0 can be configured by software as data or program memory. The DARAM can be configured in one of two ways:

i. All 1056 words 16 bits configured as data memory

ii. 544 words 16 bits configured as data memory and 512 words.16 bits configured as program memory

DARAM improves the operational speed of the 'C5x CPU. The CPU operates with a 4-deep pipeline. In this pipeline, the CPU reads data on the third stage and writes data on the fourth stage. Hence, for a given instruction sequence, the second instruction could be reading data at the same time the first instruction is writing data. The dual data buses (DB and DAB) allow the CPU to read from and write to DARAM in the same machine cycle.

DATA/PROGRAM SINGLE-ACCESS RAM :

All 'C5x DSPs except the 'C52 carry a 16-bit on-chip single-access RAM (SARAM) of various sizes . Code can be booted from an off-chip ROM and then executed at full speed, once it is loaded into the on-chip SARAM. The SARAM can be configured by software in one of three ways:

i. All SARAM configured as data memory

ii. All SARAM configured as program memory

iii. SARAM configured as both data memory and program memory

The SARAM is divided into 1K- and/or 2K-word blocks contiguous in address memory space. All 'C5x CPUs support parallel accesses to these SARAM blocks. However, one SARAM block can be accessed only once per machine cycle. In other words, the CPU can read from or write to one SARAM block while accessing another SARAM block. When the CPU requests multiple accesses, the SARAM schedules the accesses by providing a not-ready condition to the CPU and executing the multiple accesses one cycle at a time.

ON-CHIP MEMORY PROTECTION :

The 'C5x DSPs have a maskable option that protects the contents of on-chip memories. When the related bit is set, no externally originating instruction can access the on-chip memory spaces.

ON-CHIP PERIPHERALS :

All 'C5x DSPs have the same CPU structure; however, they have different on-chip peripherals connected to their CPUs. The 'C5x DSP on-chip peripherals available are:

a) Clock generator
b) Hardware timer
c) Software-programmable wait-state generators
d) Parallel I/O ports
e) Host port interface (HPI)
f) Serial port
g) Buffered serial port (BSP)
h) Time-division multiplexed (TDM) serial port
i) User-maskable interrupts

CLOCK GENERATOR :

The clock generator consists of an internal oscillator and a phase-locked loop (PLL) circuit. The clock generator can be driven internally by a crystal resonator circuit or driven externally by a clock source. The PLL circuit can generate an internal CPU clock by multiplying the clock source by a specific factor, so you can use a clock source with a lower frequency than that of the CPU.

HARDWARE TIMER :

A 16-bit hardware timer with a 4-bit prescaler is available. This programmable timer clocks at a rate that is between 1/2 and 1/32 of the machine cycle rate (CLKOUT1), depending upon the timer's divide-down ratio. The timer can be stopped, restarted, reset, or disabled by specific status bits.

SOFTWARE-PROGRAMMABLE WAIT-STATE GENERATORS :

Software-programmable wait-state logic is incorporated in 'C5x DSPs allowing wait-state generation without any external hardware for interfacing with slower off-chip memory and I/O devices. This feature consists of multiple wait-state generating circuits. Each circuit is user-programmable to operate in different wait states for off-chip memory accesses.

PARALLEL I/O PORTS :

A total of 64K I/O ports are available, sixteen of these ports are memory-mapped in data memory space. Each of the I/O ports can be ad-dressed by the IN or the OUT instruction. The memory-mapped I/O ports can be accessed with any instruction that reads from or writes to data memory. The IS signal indicates a read or write operation through an I/O port. The 'C5x can easily interface with external I/O devices through the I/O ports while requiring minimal off-chip address decoding circuits.

HOST PORT INTERFACE (HPI) :

The HPI available on the 'C57S and 'LC57 is an 8-bit parallel I/O port that pro-vides an interface to a host processor. Information is exchanged between the DSP and the host processor through on-chip memory that is accessible to both the host processor and the 'C57.

SERIAL PORT :

Three different kinds of serial ports are available: a general-purpose serial port, a time-division multiplexed (TDM) serial port, and a buffered serial port (BSP). Each 'C5x contains at least one general-purpose, high-speed synchronous, full duplexed serial port interface that provides direct communication with serial devices such as codecs, serial analog-to-digital (A/D) converters, and other serial systems. The serial port is capable of operating at up to one-fourth the machine cycle rate (CLKOUT1). The serial port transmitter and receiver are double-buffered and individually controlled by maskable external interrupt signals. Data is framed either as bytes or as words.

BUFFERED SERIAL PORT (BSP) :

The BSP available on the 'C56 and a ‘C57 device is a full-duplexed, double-buffered serial port and an auto buffering unit (ABU). The BSP provides flexibility on the data stream length. The ABU supports high-speed data transfer and reduces interrupt latencies.

TDM SERIAL PORT :

The TDM serial port available on the 'C50, 'C51, and 'C53 devices are a full-duplexed serial port that can be configured by software either for synchronous operations or for time-division multiplexed operations. The TDM serial port is commonly used in multiprocessor applications

USER-MASKABLE INTERRUPTS:

Four external interrupt lines (INT1±INT4) and five internal interrupts, a timer interrupt and four serial port interrupts, are user maskable. When an interrupt service routine (ISR) is executed, the contents of the program counter are saved on an 8-level hardware stack, and the contents of eleven specific CPU registers are automatically saved (shadowed) on a 1-level-deep stack. When a return from interrupt instruction is executed, the CPU registers' contents are restored.

RESULT

Thus the architecture of TMS320C50 processor was studied.

Ex. no: 9 A	**ADDITION USING DIRECT ADDRESSSING MODE**
Date:	

AIM:

To write an assembly language program to perform addition of two numbers using direct addressing mode.

APPARATUS REQUIRED:

1. Personal computer
2. RS232C interface cable
3. TMS320C5416 DSP Processor

ALGORITHM:

- Set the data memory locations of Input 1, Input 2 and Output
- Start the program
- Initialize the data memory page.
- Reset the Complier mode (CPL) bit.
- Delay is given by NOP (No Operation).
- The data which is in data memory (A000H) is located to Accumulate A.
- The data which is in data memory (A001H) is added with Accumulator A.
- Accumulator A Result is stored at A002H.
- Halt the program.

PROGRAM:

```
INP1    .SET 0H
INP2    .SET 1H
OUT     .SET 2H
.mmregs
.text
START:
        LD   #140H,DP
        RSBX   CPL
        NOP
        NOP
        NOP
        NOP
        LD       INP1,A
        ADD    INP2,A
        STL      A,OUT
HLT:   B HLT
```

INPUT:

Data Memory:

```
A000h  0004
A001h  0004
```

OUTPUT:

Data Memory:

```
A002h  0008
```

RESULT:

Thus the Assembly language program for addition of two numbers using direct addressing mode was performed and implemented using TMS320C5416 DSP processor

Ex. no: 9 B	**SUBTRACTION USING DIRECT ADDRESSSING MODE**
Date:	

AIM:

To write an assembly language program to perform subtraction of two numbers using direct addressing mode.

APPARATUS REQUIRED:

1. Personal computer
2. RS232C interface cable
3. TMS320C5416 DSP Processor

ALGORITHM:

- Set the data memory locations of Input 1, Input 2 and Output
- Start the program
- Initialize the data memory page.
- Reset the Complier mode (CPL) bit.
- Delay is given by NOP (No Operation).
- The data which is in data memory (A000H) is located to Accumulate A.
- The data which is in data memory (A001H) is subtracted with Accumulator A.
- Accumulator A Result is stored at A002H.
- Halt the program.

PROGRAM:

```
INP1 .SET 0H
INP2 .SET 1H
OUT .SET 2H
.mmregs
.text
START:
        LD #140H,DP
        RSBX CPL
        NOP
        NOP
        NOP
        NOP
        LD INP1,A
        SUB INP2,A
        STL A,OUT
HLT:   B HLT
```

INPUT:

DATA MEMORY:

A000h 0004
A001h 0002

OUTPUT:

DATA MEMORY:

A002h 0002

RESULT:

Thus the Assembly language program for substraction of two numbers using direct addressing mode was performed and implemented using TMS320C5416 DSP processor.

Ex. no: 9. C	DIVISION USING DIRECT ADDRESSSING MODE
Date:	

AIM:

To write an assembly language program to perform division of two numbers using direct addressing mode.

APPARATUS REQUIRED:

1. Personal computer
2. RS232C interface cable
3. TMS320C5416 DSP Processor

ALGORITHM:

- Set the data memory locations of Dividend, Divisor, Quotient and Remainder.
- Start the program
- Initialize the data memory page.
- Reset the Complier mode (CPL) bit and fractional mode (FRCT) bit.
- Delay is given by NOP (No Operation).
- Dividend data is loaded into Accumulator A.
- Data which is in Accumulator A is divided by divisor and the result is stored in Accumulator A.
- Lower 16 bit Result (Quotient) of Accumulator A is stored at A002H.
- Higher 16 bit Result (Remainder) of Accumulator A is stored at A003H.
- Halt the program.

PROGRAM:

```
DIVID .SET 0H
DIVIS .SET 1H
QOUT .SET 2H
REMAIN .SET 3H
.mmregs
.text
START:
        STM #140H,ST0
        RSBX CPL
        RSBX FRCT
        NOP
        NOP
        NOP
        NOP
        LD DIVID,A
        RPT #0FH
        SUBC DIVIS,A
        STL A,QOUT
        STH A,REMAIN
HLT:   B HLT
```

INPUT

DATA MEMORY

A000H 0009
A001H 0002

OUTPUT

DATA MEMORY

A002H 0004
A003H 0001

RESULT:

Thus the Assembly language program for division of two numbers using direct addressing mode was performed and implemented using TMS320C5416 DSP processor.

Ex. no: 9. D

Date:

ADDITION USING INDIRECT ADDRESSSING MODE

AIM:

To write an assembly language program to perform addition of two numbers using indirect addressing mode.

APPARATUS REQUIRED:

1. Personal computer
2. RS232C interface cable
3. TMS320C5416 DSP Processor

ALGORITHM:

- Start the program
- Clear the Accumulator A
- Indirect address 1000H is assigned to auxiliary register AR4.
- Indirect address 1000H is assigned to auxiliary register AR5.
- Indirect address 3000H is assigned to auxiliary register AR6.
- Data which is in 1000H is loaded to Accumulator A.
- Data which is in 2000H is added with Accumulator A.
- Result in Accumulator A is stored at 3000H.
- Halt the program.

PROGRAM:

```
.mmregs
.text
START:
        LD #00H,A
        STM #1000H,AR4
        STM #2000H,AR5
        STM #3000H,AR6
        LD *AR4,A
        ADD *AR5,A
        STL A,*AR6+
HLT: B HLT
```

INPUT:

DATA MEMORY:

1000h 0002
2000h 0004

OUTPUT:

DATA MEMORY:

3000h 0006

RESULT:

Thus the Assembly language program for addition of two numbers using indirect addressing mode was performed and implemented using TMS320C5416 DSP processor.

Ex. no: 9. E	SUBTRACTION USING INDIRECT ADDRESSSING MODE
Date:	

AIM:

To write an assembly language program to perform subtraction of two numbers using direct addressing mode.

APPARATUS REQUIRED:

1. Personal computer
2. RS232C interface cable
3. TMS320C5416 DSP Processor

ALGORITHM:

- Start the program
- Clear the Accumulator A
- Indirect address 1000H is assigned to auxiliary register AR4.
- Indirect address 1000H is assigned to auxiliary register AR5.
- Indirect address 3000H is assigned to auxiliary register AR6.
- Data which is in 1000H is loaded to Accumulator A.
- Data which is in 2000H is subtracted from Accumulator A.
- Result in Accumulator A is stored at 3000H.
- Halt the program.

PROGRAM:

```
.mmregs
.text
START:
        LD #00H,A
        STM #1000H,AR4
        STM #2000H,AR5
        STM #3000H,AR6
        LD *AR4,A
        SUB *AR5,A
        STL A,*AR6+
HLT:   B HLT
```

INPUT:

DATA MEMORY:

1000h 0008
2000h 0003

OUTPUT:

DATA MEMORY:

3000h 0005

RESULT:

Thus the Assembly language program for subtraction of two numbers using indirect addressing mode was performed and implemented using TMS320C5416 DSP processor.

Ex. no: 9. F	**MULTIPLICATION USING DIRECT ADDRESSSING MODE**
Date:	

AIM:

To write an assembly language program to perform multiplication of two numbers using indirect addressing mode.

APPARATUS REQUIRED:

1. Personal computer
2. RS232C interface cable
3. TMS320C5416 DSP Processor

ALGORITHM:

- Start the program
- Clear the Accumulator A
- Load the Data pointer.
- Load the First and second input.
- Perform the multiplication .
- Store the result in Accumulator A.
- Halt the program.

PROGRAM:

```
.mmregs
.text
START:
        STM #0140H,ST0
        STM #40H,PMST
        STM #0A000H,AR0
        ST #1H,*AR0
        LD *AR0+,T
        ST #2H,*AR0
        MPY *AR0+,A
        STL A,*AR0
HLT:    B HLT
.END
```

RESULT:

Thus the Assembly language program for multiplication of two numbers using indirect addressing mode was performed and implemented using TMS320C5416 DSP processor.

Ex. no: 10	**GENERATION OF VARIOUS SIGNALS AND RANDOM NOISE**
Date:	

AIM:

To write a program to generate various signals and random noise using TMS320C54 processor.

APPARATUS REQUIRED:

1. Personal computer
2. RS232C interface cable
3. TMS320C5416 DSP Processor

PROCEDURE:

- Desktop →Click CCStudio(Code Composer) 3.1.
- Project →New→ Project Name
- File→ New →Source file →Type the Program
- File→ Save the program into corresponding project folder →files of type .c file
- Project →Right click →Add files to project →Open the c file in corresponding folder
- Project→ Right click→ Add files to project →Open micro5416.cmd file in the syllabus folder
- Project →Rebuild all
- Desktop →Click Vi debugger software
- Serial→ Port Settings →Autodetect→ Vi dsp hardware found in Com1→ ok
- Tools →Convert .out to .asc →Open the project folde→r Debug open .out file →ok
- Serial→ Load Program →browse file→ Ok
- Serial→ Communication Window→ Give the input in the corresponding address
- To execute the file Type →GO 0000
- To display the result →Execute after successful download.

PROGRAM:

To Generate Square Wave

```
ioport int port4;
main()
{
int i,x;
while(1)
{
for(i=0;i<=0x5ff;i++)
{
x=0x0000;
port4=x;
}
for(i=0;i<=0x5ff;i++)
{
x=0xfff;
port4=x;
}
}
}
```

To Generate Saw Tooth Wave

```
ioport int port4;
main()
{
int i,x;
while(1)
{
x=0x0;
for(i=0;i<=0xfff;i++)
{
port4 = x;
x=x+5;
}
}
}
```

To Generate Triangle Wave

```
ioport int port4;
main()
{
int i,x;
while(1)
{
x=0x000;
for(i=0;i <= 0xfff;i++)
{
port4 = x;
x=x+1;
}
for(i=0;i <= 0xfff;i++)
{
port4 = x;
x=x-1;
}
}
}
```

RESULT:

Thus the program for generation of various signals and random noise was executed successfully & implemented using TMS320C5416 Processor.