Department of Computer Science and Engineering

Kumaraguru College of Technology

COIMBATORE - 641 006

CERTIFICATE

This is to Certify that the Report entitled

Implementation of Digital Filters Using TMS 32010

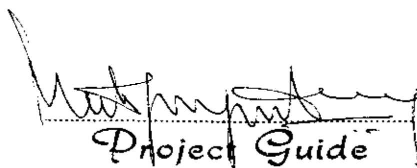
has been Submitted by

Mr. E. Jaganathan, R. [illegible]

in partial fulfilment for the award of

Bachelor of Engineering in the Computer Science and Engineering Branch of the Bharathiar University, Coimbatore-641 046

during the academic year 1993-1994



Project Guide

Head of the Dept.

Certified that the candidate was examined by us in project work

Viva-Voce Examination held on ....

Register Number was ....

Internal Examiner

External Examiner

Dedicated to our Beloved
Parents & Teachers

Acknowledgement

ACKNOWLEDGEMENT

It gives us a great pleasure to express our gratitude to **Dr.S. Subramanian**, B.E., M.Sc (Engg.) Ph.D., SMIEEE (USA) Principal, KCT and the management for the interest and encouragement given in making this project a success.

We are obliged to **Prof. P.Shanmugam** B.E. , M.Sc(Engg), M.S (Hawaii),MISTE, SMIEEE (USA), Head Of Dept, CSE, KCT who has been especially enthusiastic in giving his impressions, constructive and critical reviews.

We are greatly indebted to our beloved guide **Mr. Muthuraman Ramswamy**, M.E., MISTE., MIE., MIEEE (USA), C.Engr., for his excellent guidance. His valuable assistance helped us a great deal during our project.

Our profound thanks are due to Mr. K.Ramprakash M.E., senior lecturer, dept. of ECE for his valuable comments on our project.

Our profused gratitude is also to the CSE staff for the encouragement given in making this project a successful one.

We are thankful to the staff of CSE lab,ECE lab for all the favours and assistance rendered to us.

We express our heartful thanks to our friends and all others who helped us in completing this project.

SYNOPSIS

This project entitled IMPLEMENTATION OF DIGITAL FILTERS USING TMS 32010, involves implementation of Infinite Impulse Response filters using the DSP chip TMS 32010.TMS 32010 is the first digital signal processor of its series developed by TEXAS INSTRUMENTS INC.

The essential feature of this project is the usage of DSP chip TMS 32010 which has a very high throughput, which is the result of the comprehensive, efficient and easily programmed instruction set and the highly pipelined architecture.

Specialized instructions have been incorporated to speed up the execution of digital signal processor algorithms.

This is a novel and sincere attempt involving the DSP processor TMS 32010.A real-time signal is captured and filtered for a specified frequency band. The filtered output can be viewed on a computer screen.

Contents

CONTENTS

PAGE NO

1. INTRODUCTION	1
2. DIGITAL SIGNAL PROCESSING	4
2.1. FROM ANALOG TO DIGITAL	4
2.2. DIGITAL FILTER CLASSIFICATIONS	8
2.3. COMPARISION BETWEEN FIR AND IIR FILTERS	15
2.4. BASIC DESIGN PARAMETERS	16
2.6. IIR FILTER DESIGN TECHNIQUES	17
2.7. INDIRECT APPROACHES FOR IIR FILTER DESIGN	18
2.7.1. BUTTERWORTH FILTERS	19
2.7.2. CHEBYSHEV FILTERS	19
2.8. IMPULSE INVARIANT TRANSFORMATION	21
2.9. THE BILINEAR Z - TRANSFORMATION	26
3. AN EXAMPLE DESIGN	29
4. TMS 32010 DETAILS AND INSTRUCTION SET SUMMARY	31
4.1. DESCRIPTION	31
4.2. KEY FEATURES OF TMS 32010/C10	32
4.2.1. KEY FEATURES	32
4.2.2. ARCHITECTURE	32
4.2.3. 32-BIT ALU/ACCUMULATOR	33

4.2.4. SHIFTERS 34
4.2.5. 16 x 16-BIT PARALLEL MULTIPLIER 34
4.2.6. DATA AND PROGRAM MEMORY 34
4.2.7. PROGRAM MEMORY EXPANSION 35
4.2.8. INTERRUPTS AND SUBROUTINES 35
4.3. INSTRUCTION SET 36
4.3.1. DIRECT ADDRESSING 37
4.3.2. INDIRECT ADDRESSING 37
4.3.3. IMMEDIATE ADDRESSING 37
5. FLOW CHARTS 39
6. PROGRAMS 44
7. OUTPUT 99
8. CONCLUSION 111
9. REFERENCES 112
10. APPENDIX 113
10.1. FLOW DIAGRAM 113
10.2. PINOUT DIAGRAM OF TMS 32010 114
10.3. BLOCK DIAGRAM OF TMS 32010 115
10.4. INSTRUCTION SET OF TMS 32010 117

INTRODUCTION

Humans communicate with the external world by processing signals.For example,

1. vision is through the processing of light.
2. hearing is through the processing of sound.

Signal processing is concerned with

1. representation,
2. transformation and
3. manipulation

of signals and the information they contain.Signals are represented mathematically as a function of time with one or more independent variables.For example, a speech signal is represented mathematically as a function of time and a photographic image is represented as an brighteness fuction of two spatial variables.

With the advent of computers, miracles are being achieved in the computational speed.Then came the idea of processing signals through computers. Since the breakthroughs are in the digital computers, there came the idea of representing and processing signals digitally.This led to the development of digital signal processing.

The digital signal processing involves highly sophisticated digital systems capable of performing complex tasks which are too dificult or/and too expensive to perform using analog circuitry.In particular digital signal processing allows programmable operations.

This project incorporates TMS 32010, the specialized digital processor developed for signal processing functions to implement digital filters.Here real time signals are captured and filtered for specified frequency bands.

Signals arise in almost every field of science and engineering eg. acoustics, biomedical engineering, communication, control systems, radar physics, seismology, telemetry etc. this general class of signals can be classified in to namely continuousand descrete time signals.

A continous signal is one that is defined at each and every instant of time . Descreate time signal on the other hand is one that is defined at descrete intervals of time.this descrete time signal can be used to simulate analog signal or to realize signal transforms that cannot be implemented with continous time hardware.

Filtering is a process by which the frecquency spectrum of a signal can be modified,reducedor manipulated according

to some desired specifications. A filtering action may remove signal noise,seperate two distinct signals purposely mixed, resolve signals into their frequency components, demodulate signals, bandlimit signals.

A digital filter is a digital system that can be used to filter descrete time signals. It can be implemented by means of software or by means of dedicated hardware. In our project we use both dedicated hardware,software and realize IIR digital filters.

The hardware contains a specialized signal processor TMS32010 equipped with an A/D and a D/A converters.This hardware can be fixed into a P.C slot that it is easy to program the chip by a cross assembler.This chip employs the Harvard architecture wherein there is a seperate data memory and a program memory. This enables manipulation of vast,real-time signals.Operations like shifts are easily done with arithmatic instructions in a single cycle.

2. DIGITAL SIGNAL PROCESSING

2.1 FROM ANALOG TO DIGITAL

Fig 1.a shows a simple first-order RC filter. The simple differential equation describing this circuit as its input and output voltages is

$$v_o(t) + RC.dv_o(t)/dt = v_i(t) \quad \text{------>} \ (1)$$

where v0(t) and vi(t) are analogue output and input voltage waveforms. In the analogue world both input and output voltages are continous-time waveforms and the complexity of the solution would depend on the input voltage function Vi(t), the soluton can be obtained using

(i) Standard mathematical techniques which solve the differential equation and obtain the output waveform in closed form.

(ii) Numerical techniques which calculate the approximate output waveform in a digital computer.

The second method above provides the basis for digital filtering techniques. Consider that the input and output voltages are sampled with a sampling interval T such that vi(nT) and v0(nT) represents the values of vi(t) and v0(t) at time t = nT.

If T is sufficiently small then the derivative dv0(t)/dt at time t = NT can be approximated by

$$dv_o(nT)/dt = (v_o(nT) - v_o((n-1)T))/T \text{ -----> (2)}$$

substituting this in equation (1) we obtain

$$v_o(nT) + RC.v_o(nT)/T - RC.v_o((n-1)T)/T = v_i(nT) \text{ ---> (3a)}$$

Equation (3a) is a linear differnce equation that approximates the differential equation (1). Equation (3a) can be rewritten as

$$v_o(nT) = v_i(nT)/(1+RC/T) + (RC/T)v_o((n-1)T)/(1+RC/T) \quad \text{(3b)}$$

This is now a recursion formula in which the present input sample and the previous output sample are used to calculate the present output sample. The notation can be simplified to

$$v_o(n) = b_o v_i(n) + a_1 v_o(n) \qquad \text{------> (4a)}$$

where $b_o = 1/(1 + RC/T)$ and $a_1 = (RC/T)/(1 + RC/T)$.

The signal-flow diagram for this filter is shown in Fig.1b.

The block labelled 'D' represents a delay equal to one samling period T. In digital filter notations a delay of n sampling periods is usually denoted by (Z ** (-n)). Therefore a delay of one samling period can be represented by (Z ** (-1)).

It is important to note that a common element in all filter structures is the concept of storage.In the analogue RC filter (Fig.1a) the storage is present in the form of a capacitor and in its digital equivalent (Fig.1b) the storage takes the form of a delay stage.In fact the storage element is the essential ingredient for any filter, whether analogue or digital. This is because filters are used to operate on the signal "changes" and as such they need to have some knowledge of the history of the signal to allow them to perform their function.

An important characteristic feature of any filter is its so called "impulse respone". this is defined as the output waveform of the filter when a unity impulse is applied to the input. Using equation (4a) and assuming a unity impulse as the input waveform, i.e.

$$v_o(0) = 1$$

$$v_o(n) = 0 \quad \text{for} \quad n > 0$$

then the output sequence would be

b_o, $a_1 b_o$, $(a_1 ** 2)b_o$,.........., $(a_1 ** n)b_o$,....

or in short

$$v_o(n) = (a_o ** n)b_1$$

It should be noted that the above impulse response has, in theory, infinite length.This is due to the recursive nature of this particular filter structure. This type of filter is often referred to as an "infinite-impulse-response" (IIR) filter.

An alternative way of looking at the filter in this is to use equation (4a) in successive substitutions ; i.e.

$$v_o(n) = b_o v_i(n) + a_1 v_o(n-1)$$

$$= b_o v_i(n) + a_1 [b_o v_i(n-1) + a_1 v_o(n-2)]$$

$$= b_o v_i(n) + a_1 b_o v_i(n-1) + a_1 **2 [b_o v_i(n-2) + a_1 v_o(n-3)]$$

$$= .... \qquad -------->(4b)$$

$$= ....$$

$$= b_o v_i(n) + a_1 b_o v_i(n-1) + a_1^{2} b_o v_i(n-2) + a_1^{3} b_o v_i(n-3)$$

The above equation expresses the output waveform as a linear combination of input samples only,but this involves an infinite number of input samples. Notice also that the coefficients b0 and a1 have positive values less than unity (R and C are assumed to be finite and non-zero). This means that in equation (4b) the coefficients decrease for older input samples. It may therefore be reasonable to assume that these coefficients approximate to zero beyond a certain point. In this only a finite number of terms would be involved in equation (4b) or, in other words, the infinite impulse response is approximated by a finite impulse response since it decades rapidly to zero. This modified filter with its finite duration impulse response falls in the category of FIR (Finite-Impulse Response) filters.

2.2 DIGITAL FILTER CLASSIFICATIONS

Linear difference equations, similar to equations (4a) and (4b) are the basis for the theory of filters.the general difference equation can be expressed as

$$y(n) + \quad a_m\, y(n-m) = \quad b_k\, x(n-k) \text{ ------->(5)}$$

where the x and y sequences are the input and output of the filter and am's and bk's are the coefficients of the filter.

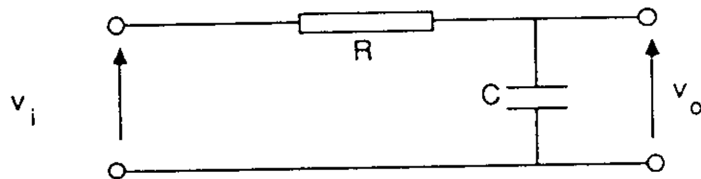
As mentioned,the notaion z is often used to denote a delay equal to one sampling peorid.In the theory of the discrete-time signals,the concept of z has been developed further and is refrred to as the z-transform.This is a discrete-time version of the well known laplace transform (sometimes referred to as the s-transform) which is mainly used for dealing with continuous signals.In the s-domain a delay of T seconds corresponds to (e**-st) .Therefore the two variables s and z are related by

$$z^{-1} = e^{-st} \quad \text{------->} \ (6)$$

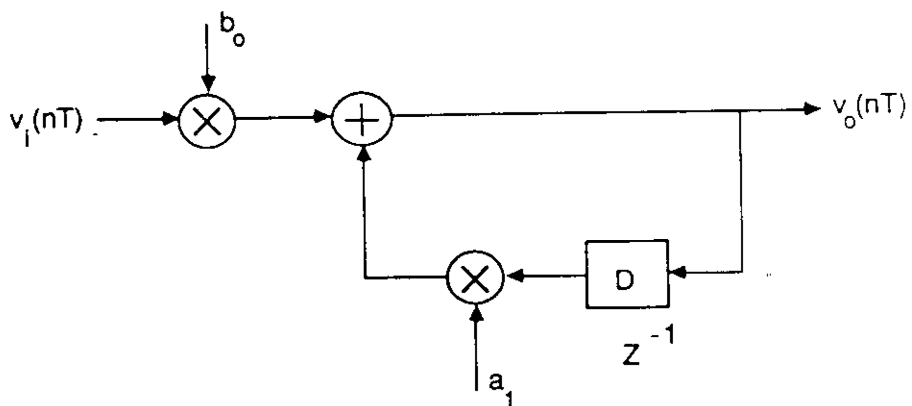
where T is the sampling period.

In the s-domain,the spectrum of a signal with a bandwidh B and sampled at a frequency fs is periodic,with a period equal to fs.This is depicted in pic2.This periodicity in the spectrum of a sampled signal is the basic reason behind the nyquist criterion which requires a minimum sampling frequency of twice the signal bandwidh(i.e.fs min = 2xB) in order to avoid aliasing effects.

Equation (6) allows a mapping between the two domains.Part of the imaginary axis between -fs/2 to +fs/2,in



(a) Analogue RC filter



(b) Discrete-time version of (a)

Fig.1 Analogue RC filter and its discrete-time equivalent.

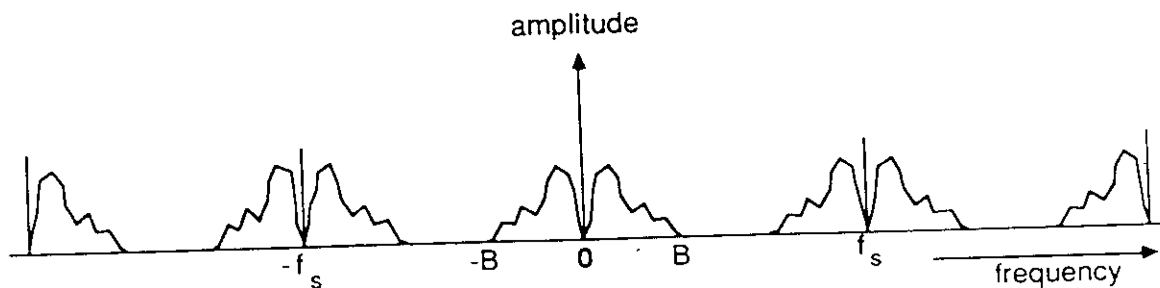


Fig.2 Spectrum of a sampled signal. B is the bandwidth of the signal and f_s is the sampling frequency.

the s-plane is mapped onto a circle is a consequence of the periodic nature of the spectrum.As shown in fig.3,the left-hand half of the s-plane(between -fs/2 and +fs/2) is mapped onto the outside of the circle.

As in the analogue design(s-domain)where a pole in the wrong place,i.e.in the right-half plane,indicates instability,in the case of discrete-time signals(z-domain) a pole outside the unit circle causes instabilities.In both cases zeros can be anywhere.

Using the z-transform notation,the general linear equation(5) can be expressed as

$$Y(z)(1+ \quad a_m z^{-m} = X(z) \quad b_k z^{-k} \qquad \text{---------->}\ (7)$$

where X(z) and Y(z) are the z-transforms of the input and output waveforms.The discrete-time(or digital) transfer function of the general filvgr is thus given by

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_k \quad z^{-k}}{1+ \quad a_m \quad z^{-m}}$$

------------->8

In terms of realization,digital filters are classified into nonrecursive and recursive types.The nonrecursive structure contains only feed-forward paths and as such all the am terms (equation(8)) are zero.This means that for the nonrecursive filters the output is the sum of a number of linearly weighted present and last samples of the input signal,as shown in fig.4.Referring to equation (8),for the nonrecursive filters the transfer function has only zeros and,as such ,is always stable.

In the recursive filters,on the other hand,some or all of the am terms are non-zero,resulting in the presence of both poles and zeros in the transfer function.Fig.5a shows the general recursive filter structure.Fig.5b shows an alternative structure for the same transfer function with a reduced number of delay stages.

Digital filters are classified in terms of their impules responses.In this classification ,those filters with a finite duration impulse response are referred to as FIR filiters and those with an infinite duration impulse response are called IIR filiters.The simplest FIR filter realization is in the nonrecursive form.For example in Fig.4,if a unit impulse is clocked through the filiter,the sequence,

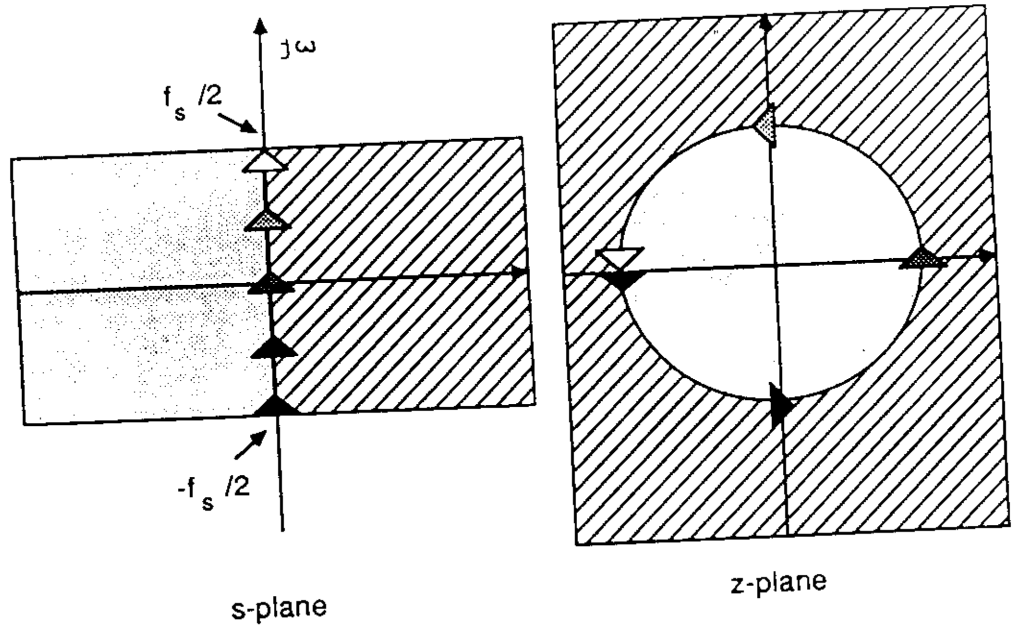


Fig.3 Relationship between the *s*-domain and the *z*-domain.

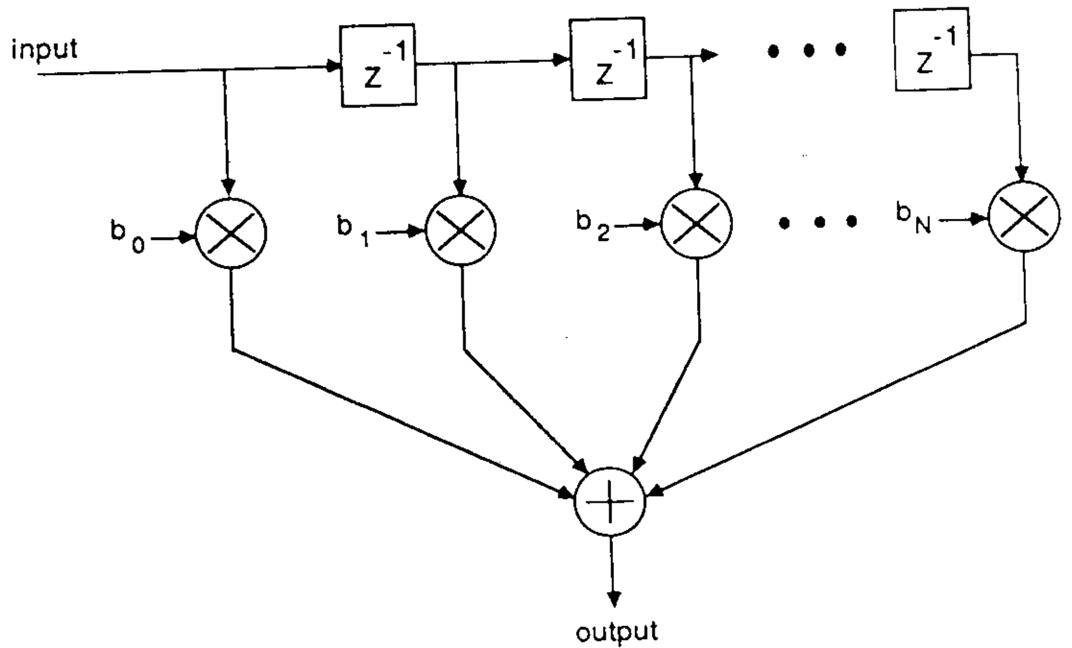


Fig.4 Nonrecursive digital filter structure.

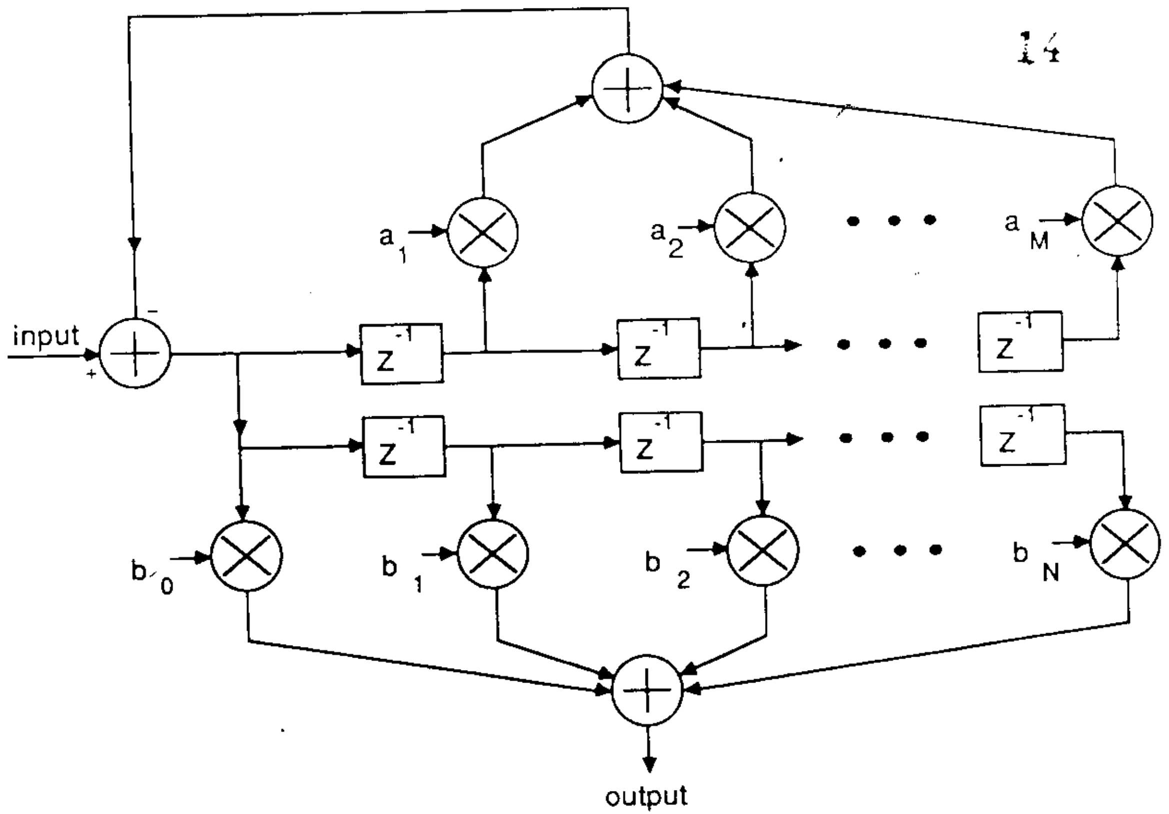


Fig.5a **Recursive (IIR) digital filter structure.**

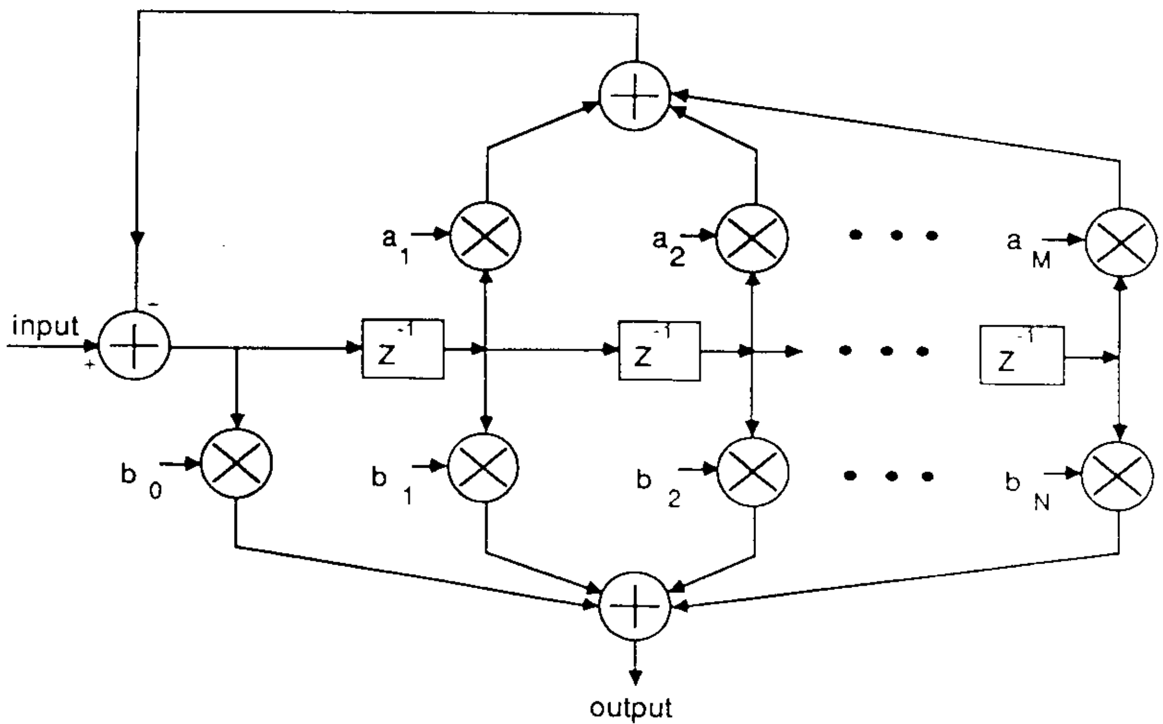


Fig.5b **Alternative recursive (IIR) digital filter structure with reduced number of delay stages.**

b0,b1,b2,.....bn,0,0,0,0,0,......0,0,0 ------>9

will be output.Notice that the response consists of a sequence of samples corresponding to the coefficients followed by zeros,i.e.the nonrecursive structure is an FIR filiter.On the other hand,the impulse response of the recursive structure(Figs.5a,5b),because of the feedfack paths,is infinite in duration,making the configuration an IIR filter

Digital filiter design methods can be divided into two categories:

(a) Design techniques suitable for FIR filiters

(b) Design techniques suitable for IIR filiters

In both cases the requirements is simply the choice of filiter coefficients in such a way that the specification for required transfer function is met.

2.3 COMPARISION BETWEEN FIR AND IIR FILTERS

FIR filters,because of their finite-impulse response,have no counterparts among analogue filters and as such can implement transfer functions which cannot be realized in the analogue world.One such property is the

excellent linear-phase characteristic,which can easily be realized with FIR filters.Since a linear-phase response corresponds to only a fixied delay,attention can be focussed on approximating the desired magnitude response without concern for the phase.The design techniques for FIR filters are generally simpler that those for IIR filters and,as there are no feedfack paths in an FIR filter.the stability of the filter is guaranteed. Also FIR filters have been employed in,and algorithms eveloped for,adaptive processing,while the use of IIR filters in these types of systems is not common.

IIR filters,on the other hand,have infinite impulse response and thus their design can be closely related to anologue filter design. IIR filters in general require fewer stages compare to FIR filters, but their stability is not unconditional and great care should be taken to ensure stability. Further more, IIR filters donot generally result in linear face charecteristics, which is important in many applications.

2.4 BASIC DESIGN PARAMETERS

For convenience, in digital filter design the frequency axis is usually normalized with respect to the sampling frequency Fs. For example for a filter with an actual pass

band cutoff frequency of 20KHz, a stop band cutoff frequency of 30KHz and a sampling frequency of 100KHz we have

The normalized pass band cutoff frequency Fpb=20/100=0.2

The normalized stop band cutoff frequency Fsb=30/100=0.3

As shown fig 6. the useful frequency axis (normalized) extends from 0.0 to 0.5, because the Nyquist sampling theorem requires a signal to be sampled at more than twice its highest frequency. This means that the ratio of the frequecy of any component in the signal to the sampling frequency must always be less than 0.5.

Reffering to fig 6., the pass band and stop band ripples are usually expresed in dBs ie,

$$\text{Pass band ripple (dB)} = 20\log_{10}(1+d_1)$$

$$\text{Stop band ripple (dB)} = 20\log_{10}(d_2)$$

The parameters Fsb Fpb, d and the sampling frequency define the basic specificaion of a filter prior to its design.

2.6 IIR FILTER DESIGN TECHNIQUES

The probelm of designing recurssive filter is one of determining the feed forward and feed back coefficients. The

design techniques for IIR filters can be categorized into two basic groups

1. Indirect approaches
2. Direct approaches

2.7 INDIRECT APPROACHES FOR THE DESIGN OF IIR FILTERS

As mentioned eairlier, digital recurssive filters are colsely related to conventional anologue filters. In the indirect method this similarity is exploited and the digital filter coefficients are determined from a suitable analogue filer, using some form of transformation technique in other words indirect apprach uses the wealth knowledge already available on anologue filters (such as Butterworth, Chebyshev and Elliptic filters) and develops a corresponding recurssive digital filter. This method involves the following two steps

1. The determination of a suitable analogue filter transrfer funcion H(s)
2. Transformation digitization of this analogue filter some of the most popular design techniques falling into the direct catogory are
   a. Impulse-invariant transformation
   b. Billnear z-transform
   c. Matched z-transform

These three techniques can be employed to derive recurssive digital filters from conventional analogue filter structures. Before discussing these three techniques, the basic characteristics of the common analogue filters, from which IIR filters are derived, will be briefly reveiwed. The starting point in the indirect IIR design techniques is often one of the following analogue filters types

2.7.1 **Butterworth filters** These filters are characterised by the property that their magnitude characteristic is maximally flat at the origin of the s-plane. Butterworth filters are specified their magnitude-square function i.e.,

$$|H(s)|^2 = 1/(1+(s/sc)^{2n} \qquad \text{------->10}$$

The pole locations in the s-plane are equally spaced around a circle of radius wc (sc=jwc). These filters have a monotonically decreasing amplitude fuction with a roll-off of approximately 6n dB/decade. Fig 7. shows the overall amplidue response of this type of filter.

2.7.2 **Chebyshev filter**

In this types of filters the peak magnitude of the approximation error is minimized over a prescribed band of frequency and also equiripple over the band. Chebyshev filters are specified by the magnitude-square function

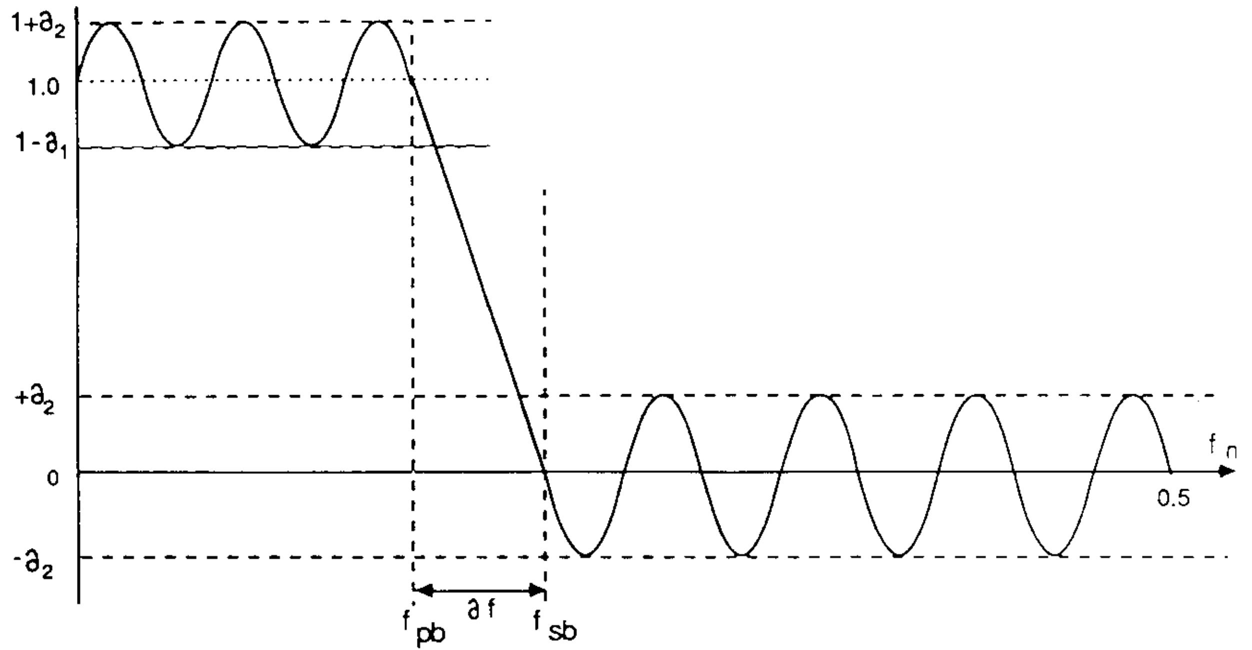


Fig.6 Specification parameters for a low pass filter. Similar parameters exist for high pass and band pass filters.

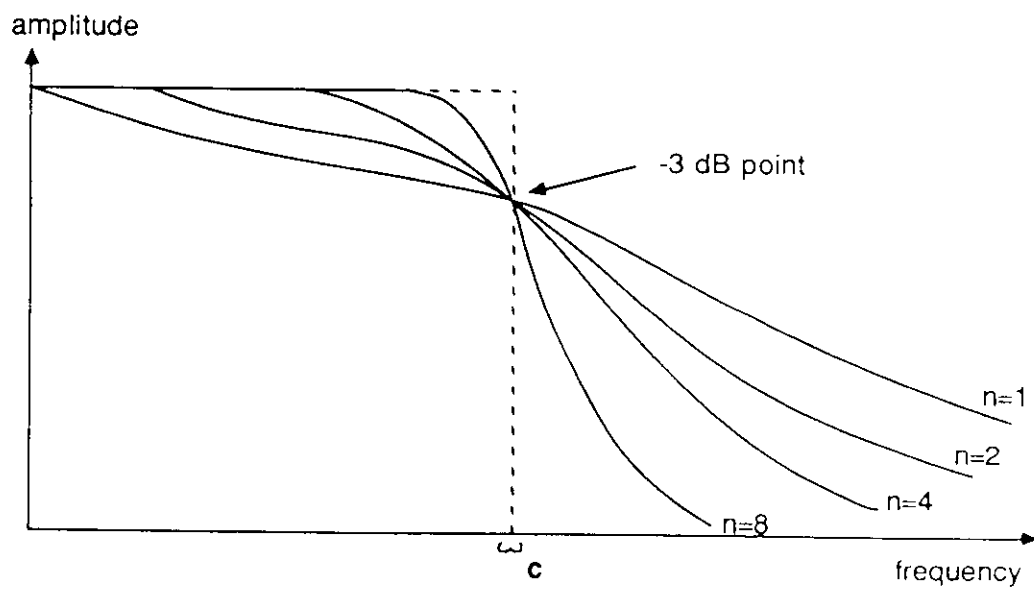


Fig. 7 Frequency response of the Butterworth filter.

$$|H(s)|^2 = 1/(1+(E^2 Cn^2 (s/sc)))$$ ------>11

where Cn(s) is a Chebishev polynomial of order n. The parameter E is used to specify a magnitude function with equal reppile in the pass band and monotony decade. Fig 8. shows the magnitude-square transfer fuction for the Chebyshev filter (type 1) where the amplitude of the repple is given by

$$d=1-(1/\ 1+E^2)$$

The poles of the Chebyshev filter lie on an ellipse determined from the parameters E,n and sc. Chebyshev filters of type 2, on the other hand, have monotony behavior in the pass band (Maximally flat around wo) and exhibit equirable behavior in the stop band.

Having decided the type and the specification of the analogue filter that satisfies the requirment, the next step in the indirect design method is to use one of the three following techniques to obtain the corresponding digital filter.

2.8 IMPULSE INVARIANT TRANSFORMATION

One of the most common techniques for driving a digital filter from a analogue filter is the impulse-invariant

transformation. As the name suggests,this technique consists of using a sampled version of the impulse response of the anaogue filter as the impulse response of the digital filter; i.e, the transformation does not change the impulse response of the analogue filter. Fig.16 illustrates the relationship between the analogue and resulting digital responses of a typical low-pass filter obtained via the impulse-invariant method. The important point to note here is that sampling the analogue signal response results in the frequency response of the resulting digital filter being periodic, with a period equal to the sampling frequency Fs. This means that the digital filter will have a frequency response similar to a repetitive version of that of the analogue filiter.If the frequency response of the analogue filter does not decay to near zero beyond fs/2 then serious alising would occur and the digital response would be corrupted.This alising problem means that this design technique is not suitable for high pass filters.However,for low-pass and band-pass filters the proplem can be avoided by choosing a sampling frequency high enough to ensure that the magnitude of the analogue filter response is negligible beyond fs/2.

To demonstrate hoe the impulse-invariant transformation is used to digitize an analogue filter,consider the simple

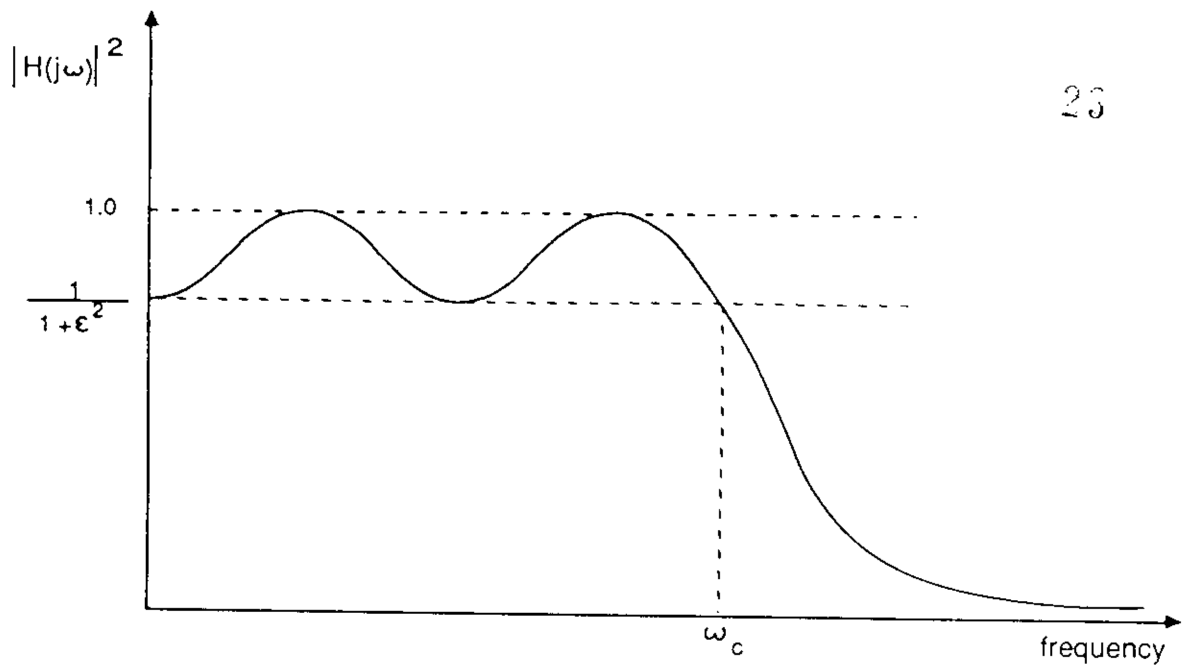


Fig. 8 **Frequency response of the Chebyshev filter (type I).**

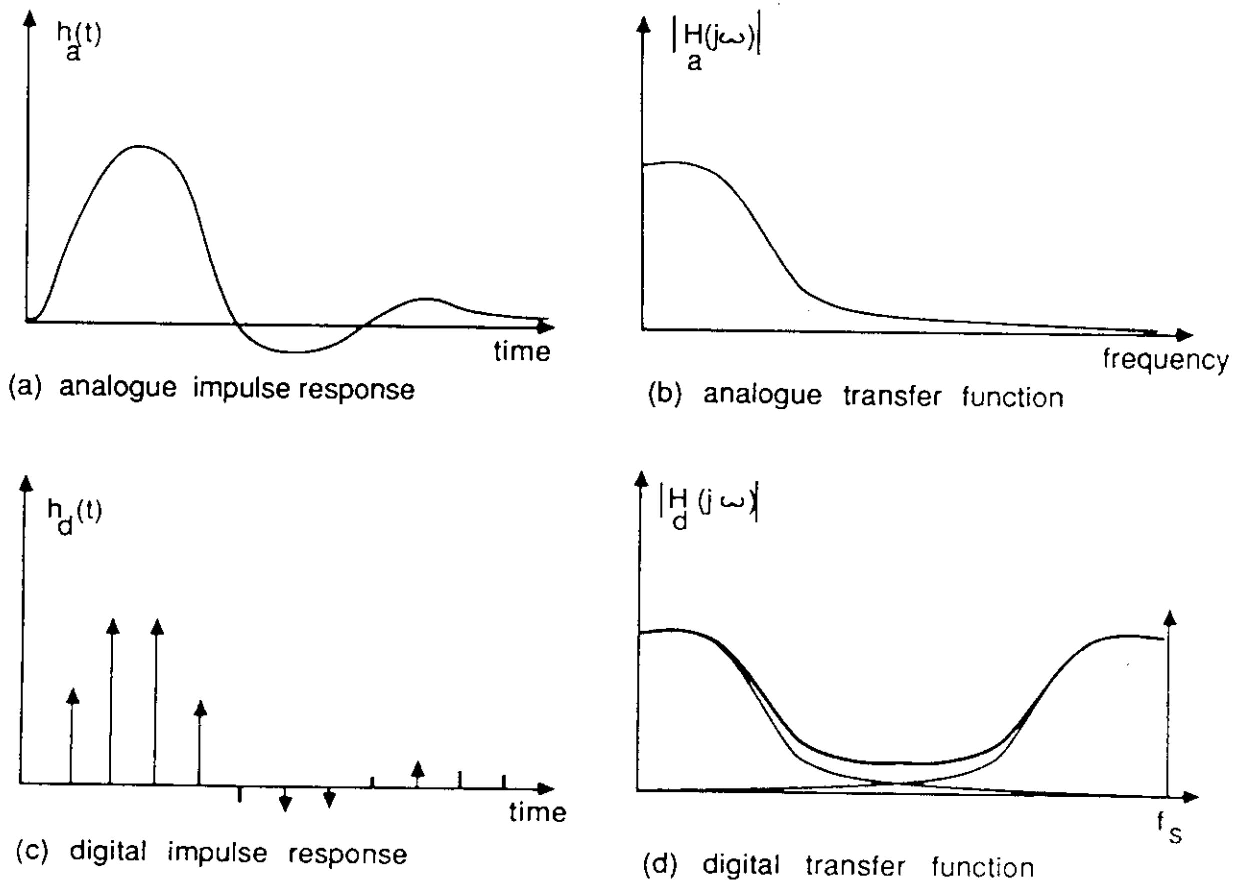


Fig. 9 **The impulse invariance transformation relationship between analogue and digital impulse and frequency respones.**

case of an analogue filter with an impulse response ha(t),i,e.a simple RC filter (the s-domain transfer function of this filter is A/s+a).We start by sampling the impulse response of this analogue filter with a sampling intreval T to obtain the corresponding impulse response for the digital filter,i.e .

$$h\ (kT)=Ae^{-\ kT} \qquad \text{------->}(12)$$

The z-transform of equation (12) is

$$H(Z)= \quad A\ e^{-\ kT}\ z^{-k} \qquad \text{------->}\ (13)$$

As equation (13) is a geometric series,the result of the summation would be

$$Hd(z) = A/(1-Z^{-1}\ e^{-\ T}) \qquad \text{------->}\ (14)$$

Equation (14) provide the z-domain transfer function of the resulting digital filter.To determine the filter coefficient(bk's and am's) equation (14) can be compared with equation (8).For this simple example it can be seen that we have

$$a1=e^{-\ T} \quad \text{and } b0 = A.$$

The impulse responses were used to arrive at the z-domain transfer function. It is more convenient to perform the impulse invariant transformation directly from the s-domain to the z-domain. It can be seen that the required mapping is of the form

$$\frac{1}{s + a} = \frac{1}{1 - e^{-kt} z^{-1}} \qquad (15)$$

As a second example, consider the two-pole analogue filter specified by

$$H_a(s) = \frac{2}{(s + 3)(s + 1)} \qquad (16)$$

$$H_a(s) = \frac{1}{s + 1} \quad \frac{1}{s + 3} \qquad (17)$$

Using equation (15), the digital transfer function would be

$$H_a(z) = \frac{(e^{-T} - e^{-3T})z^{-1}}{1 - (e^{-T} + e^{-3T})z^{-1} + e^{-4T} z^{-2}} \qquad (18)$$

Comparing equation (18) with (8)

$$b_0 = 0 \qquad b_1 = e^{-T} - e^{-3T}$$

$$a_1 = -(e^{-T} + e^{-3T}) \qquad a_2 = e^{-4T}. \qquad (19)$$

2.9 The billinear z-transformation

Another indirect design method commonly used for recursive filters is the bilinear z-transformation. The major characterisc of this transformation is that it avoids the aliasing problem which was inherent in the impulse-invariant transformation. Given as analogue transfer function H(s), let us rename the variable S to S_a to indicate the reference to the analogue world; i.e. $H(S) = H(S_a)$. Now let us define a new variable S_d related to S_a by the mapping

$$S_a = j\frac{2}{T} \tan\left(\frac{S_d T}{2j}\right) \qquad (20)$$

where T is the sampling period.

Since the analogue frequency variable w_a is related to the s-plane variable by $s_a = jw_a$, we can also express the above mapping as

$$w_a = \frac{2}{T} \tan\left(\frac{w_d T}{2}\right) \qquad (21)$$

where w_d is defined as $s_d = jw_d$.

Starting from an analogue transfer function H (jw_a), Fig. 10 illustrates the effect of this mapping on this transfer function. It can be seen from this diagram that the bilinear transformation compresses the entire analogue frequency range (w_a = 0 ---->) into a finite range equal to half the sampling frequency. This means that the spectral folding problem is completely eliminated and aliasing is therefore avoided. This compression of analogue frequency axis is usually referred to as frequency warping.

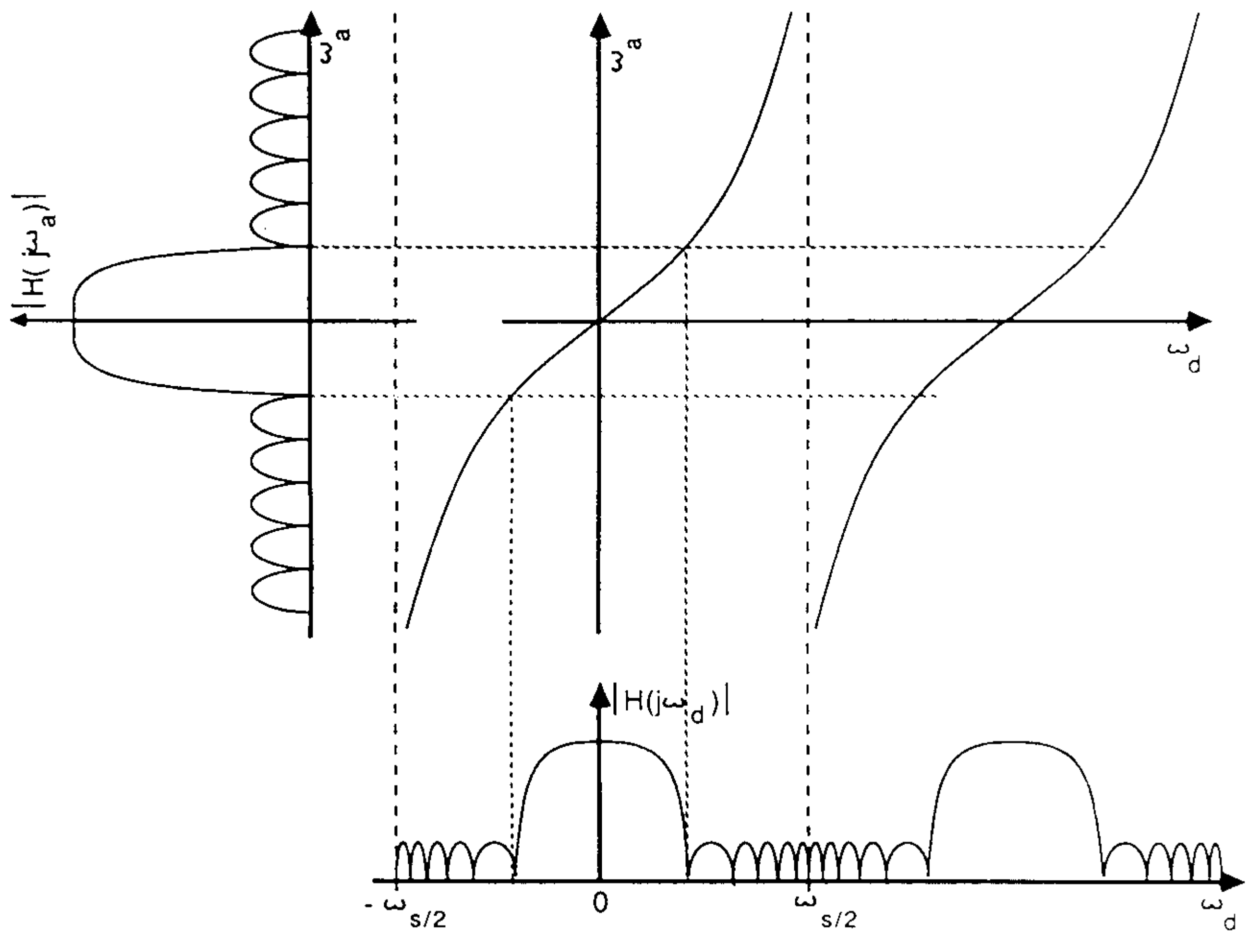


Fig. 10 Graphical illustration of the bilinear z-transform.

The price that is paid for this advantage is a distorted digital frequency scale resulting from this frequency warping. It can be seen from Fig. 10 that, due to the non-linear mapping, the specification of the resulting filter, such as the cut-off frequency, would be somewhat different from the starting analogue filter. This distortion can be taken into account in the course of digital filter design. For example, the cut-off frequency of the origional analogue filters are modified slightly so that, after mapping, the resulting filter has the desired cut-off

Returning to the transformation equation (20), we can rewrite it as

$$s_a = \frac{2}{T} \frac{1 - e^{-S_a T}}{1 + e^{-S_d T}} \qquad (22)$$

and remembering that $Z^{-1} = e^{-S_d T}$ we can write

$$S_a = \frac{2}{T} \frac{1 - Z^{-1}}{1 + Z^{-1}} \qquad (23)$$

Equation (23) provides the means for bilinear transformation directly from the S-domain to the Z-domain, suitable for digital filter implementation.

3. AN EXAMPLE DESIGN

Filter spection

low pass BUTTERWORTH 0 -->10khz pass band

order 2

sampling rate 100khz

transition band 10khz to 20khz

stop-band attenuation -10dB (starting at 20khz)

Design

Digital filter cut-off frequency = wcd= 2*3.141*10000

start of digital filter stop band = wsd=2*3.141*20000.

since the sampling rate is 100khz,the sampling period would be

T = (10 ** (-5))

therefore

wcdT = 0.2(22/7) and wsdT = 0.4(22/7)

using equation ( ), we can calculate the corresponding analogue filter frequencies, i.e.

analogue filter cut-off frequency (wca)

wca = (2/T) tan (0.1 * 3.14) = 0.6498 * 10 ** 5

start of analogue filter stop band (wsa)

wsa = (2/T) tan (0.2 * 3.14) = 1.4531 * 10 ** 5

A second order butterworth filter with a cut-off at

wca = 0.650 * 10 ** 5 has two equally spaced poles on a circle of radius wca given by

s1,s2 = -0.6498 * 10 ** 5(0.7071 + Or - 0.7071j)

= -0.4595(1.0 + or - j) * 10 ** 5

and the transfer function is given by

$$H(s) = \frac{s1s2}{(s - s1)(s - s2)} = \frac{4.223 * 10 ** 9}{s**2+0.919*10**5+4.223*10**9}$$

Now we apply the bilinear z-transforations by substituting for s in the above transfer function from equation( ). This gives the digital filter transfer function

$$H(z) = \frac{0.0675 + 0.1349*z**(-1) + 0.0675*z**(-2)}{1 - 1.1430*z**(-1) + 0.4128*z**(-2)} \quad (a)$$

The digital filter coefficients can be obtained by comparing equation (a) with (8), giving

b0 = 0.0675   a1 = -1.143
b1 = 0.1349   a2 = 0.4128
b2 = 0.0675

These coefficient values are then expressed in Q format, with a number of bits governed by the required accuracy.

4 TMS32010 DETAILS AND INSTRUCTION SET SUMMARY

4.1 DESCRIPTION :

The TMS 32010 is a 16/32 bit single-chip digital signal processor that combines the flexibility of a high speed controller with the numerical capability of an array processor, thereby offering an inexpensive alternative to multichip bit-slice processor. The highly paralleled architecture and efficient instruction set, provide speed and flexibility capable of executing 6.4 MIPS (Million Instruction Per Second). The TMS 32010 optimizes speed by implementing functions in hardware that other processors implement through microcode or software.This hardware-intensive approach provides the design engineer with processing power previously unavailable on a single chip.

The TMS 32010 is the first digital signal processor in the TMS 32010 family, was introduced in 1983 by TEXAS instruments, its powerful instruction set, inherent flexibility, high-speed number-crunching capabilities, and innovative architecture have made this high-performance, cost-effective processor the ideal solution to many telecommunications, computer, commercial, industrial, and military applications.

4.2 KEY FEATURES OF TMS32010/C10 ;

A simple block diagram showing the under mentioned keyfeatures of TMS 32010 is given in figure 1.

4.2.1 KEY FEATURES :

Instruction cycle timing

-160 ns (TMS32010-25/C10-25)

-200 ns (TMS32010/C10)

-280 ns (TMS32010-14)

* 144 Words of on-chip Data RAM
* 1.5K Words of on-chip program ROM
* External Memory Expansion up to 4k Words at Full speed
* 16 x 16 Bit Multiplier with 32-Bit product
* On-chip clock oscillator
* Single 5-v Supply

4.2.2 ARCHITECTURE :

A detailed Architectural block diagram of TMS32010 is illustrated in figure 2. The TMS32010 utilizes a modified Harvard architecture for speed and flexibility. In a strict Harvard architecture, program and data memory are in two seperate spaces, permitting a full overlap of instruction

fetch and execution. The TMS320 family's modification of the harvard architecture allows transfer between program and data spaces, thereby increasing the flexibility of the device. The modification permits coefficients stored in program memory to be read into the RAM, eliminating the need for a separate coefficient ROM. It also makes available immediate instructions and subroutines based on computed values.

4.2.3 **32-bit ALU/ACCUMULATOR :**

The TMS32010 contains a 32-bit ALU and accumulator for support of double-precision, two's complement arthimetic. The ALU is a general purpose arithmetic unit that operates on 16-bit words taken form the data RAM or derived from immediate instructions. In addition to the usual arithmetic instructions, the ALU can perform boolean operations, providing the big manipulation ability required of a high-speed controller. The accumulator stores the output from the ALU and is often an input to the ALU. It operates with a 32-bit wordlength. The accumulator is divided into a high-order word (bits 31 through 16) and a low-order word (bits 15 through 0). Instructions are provided for storing the high-and low-order accumulator words in memory.

4.2.4 SHIFTERS:

Two shifters are available for manipulating data. The ALU parallel shifter performs a left-shift of 0 to 16 places on data memory words loaded into the ALU. This shifter extends the high-order bit of the data word and zero-shift of 0, 1 or 4 places on the entire accumulator and places the resulting high-order accumulator bits into data RAM. Both shifters are useful for scaling and bit extraction.

4.2.5 16 x 16-BIT PARALLEL MULTIPLIER :

The multiplier performs a 16 x 16-bit two's complement multiplication with a 32-bit result in a single instruction cycle. The multplier consists of three units: the T-register, P-register, and multiplietr array. The 16-bit T-register temporarily stores the multiplicand; the P-register stores the 32-bit product. Multiplier values either come from the data memory or are derived immediately form the MPYK (multiply immediate) instruction word. The fast on-chip multiplier allows the device to perform fundamental operations such as convolution, correlation, and filtering.

4.2.6 DATA AND PROGRAM MEMORY :

Since the TMS32010 device use a Harvard architecture, data and program memory reside in two separate spaces. This

first-generatiuon device has 144 words of on-chip data RAM and 1.5 k words of on-chip program ROM.

4.2.7 PROGRAM MEMORY EXPANSION :

THE TMS32010 device is capable of executing up to 4k words of external memory at full speed for those applications requiring the external program memory space. This allows for external RAM-based systems to provide multiple functionality.

The TMS32010/C10 offers two modes of operation defined by the state of the MC/MP pin the microcomputer mode (MC/MP = 1) or the microprocessor mode (MC/MP = o). In the microcomputer mode, on-chip ROM iS mapped into the memory space with upto 4k words of memory available. In the microporcessor mode all 4k words of memory are external.

4.2.8 INTERRUPTS AND SUBROUTINES :

The TMS32010 contains a four-level hardware stack for saving the contents of the program counter during interrupts and subroutine calls. Instructions are .pa available for saving the device's complete context. PUSH and POP instructions permit a level of nesting restricted only by the amount of available RAM. The interrupts used in these devices are maskable.

INPUT OUTPUT :

The 16-bit parallel data bus can be utilised to perform I/O functions in two cycles. The I/O ports are addressed by the three LSBs on the address lines. In addition, a polling input for bit test and jump operations (BIO) and an interrupt pin (INT) have been incorporated for multitasking.

4.3 INSTRUCTION SET :

A Comprehensive instruction set supports both numeric intensive operations, such as signal processing, and general purpose operations, such as high-speed control. The TMS32010 feature a powerful count of 60 instructions. the instructions, permitting execution rates of more than six million instructions per second. Only infrequently used branch and I/O instructions are multicycle. Instructions that shift data as part of an arithmetic operation execute in a single cycle and are useful for scaling data in paralel with other operations.

Three main addressing modes are available with the instruction set: direct, indirect, and immediate addressing.

4.3.1 **DIRECT ADDRESSING :**

In direct addressing, seven bits of the instruction word concatenated with the 1-bit data page pointer form the data memory address. This implements a paging scheme in which the first page contains 128 words, and the second page contains 16 words.

4.3.2 **INDIRECT ADDRESSING :**

Indirect addressing forms the data memory address from the least-significant eight bits of one of the two auxiliary registers. AR0 and AR1. The auxiliary register pointer (ARP) selects the current auxiliary register. The auxiliary registers can be automatically incremented or decremented and the ARP changed in parallel with the execution of any indirect instruction to permit single-cycle manipulation of data tables. Indirect addressing can be used with all instructions requiring data operands, except for the immediate operand instructions.

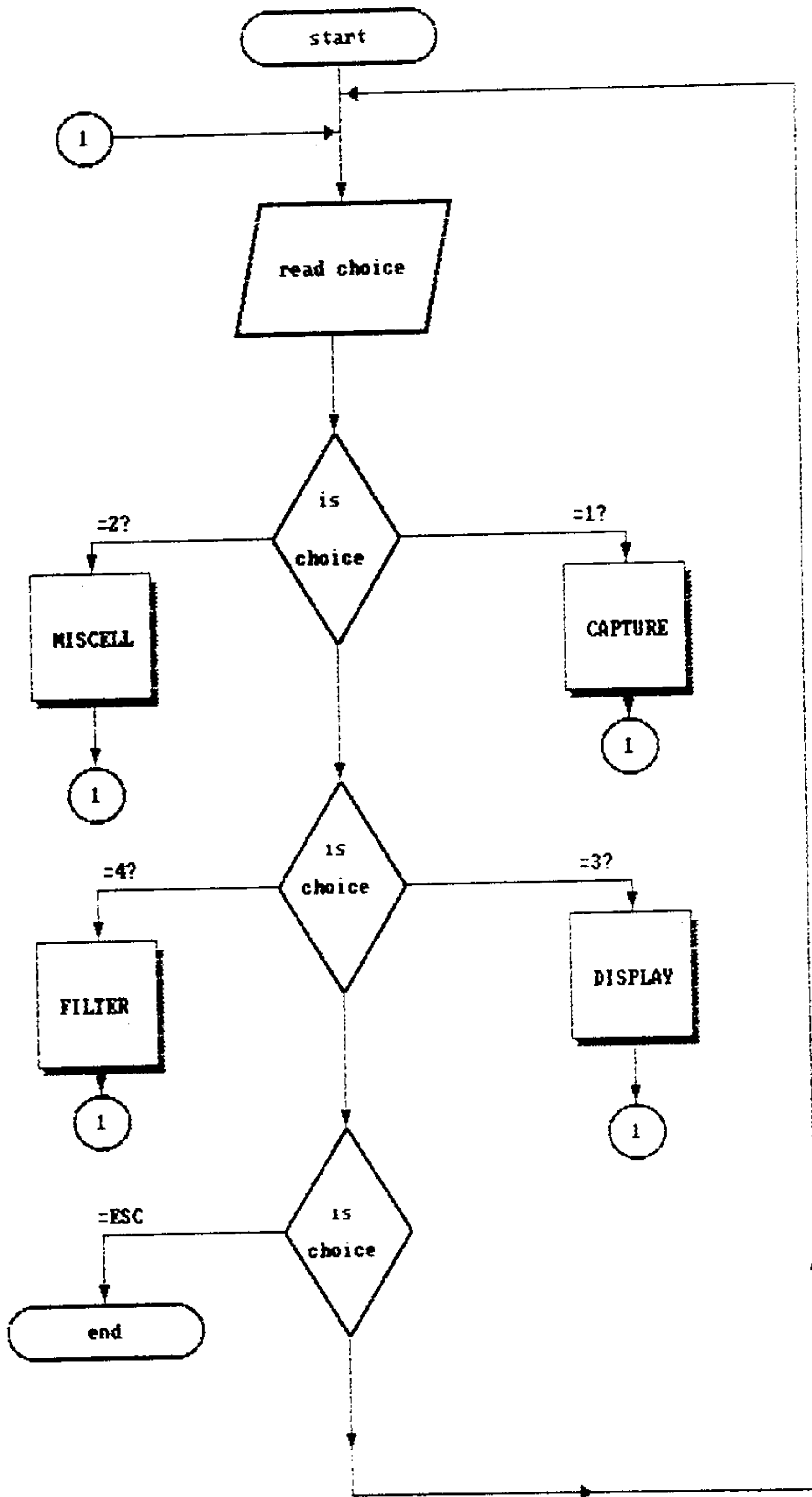
4.3.4 **IMMEDIATE ADDRESSING :**

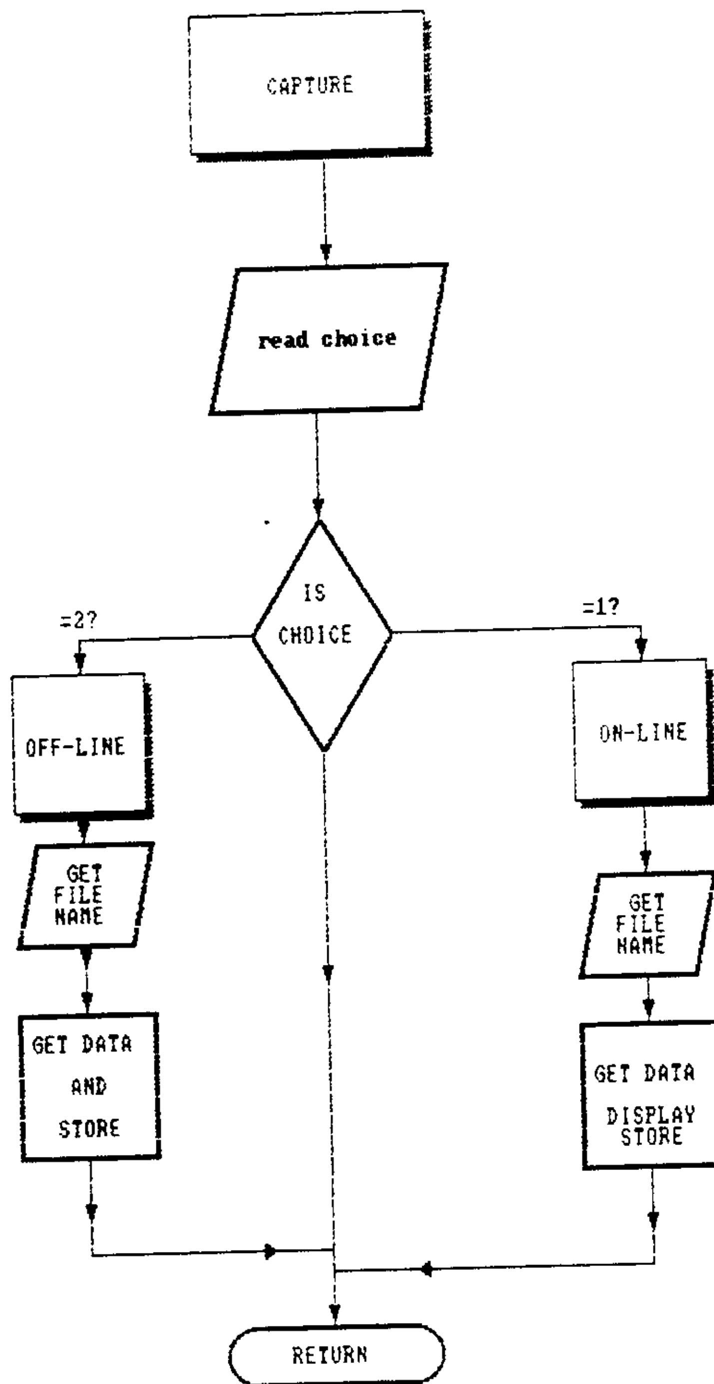
Immediate instructions derive data from part of the instruction word rather than from the data RAM. Some useful immediate instructions are multiply immediate (MPYK), load

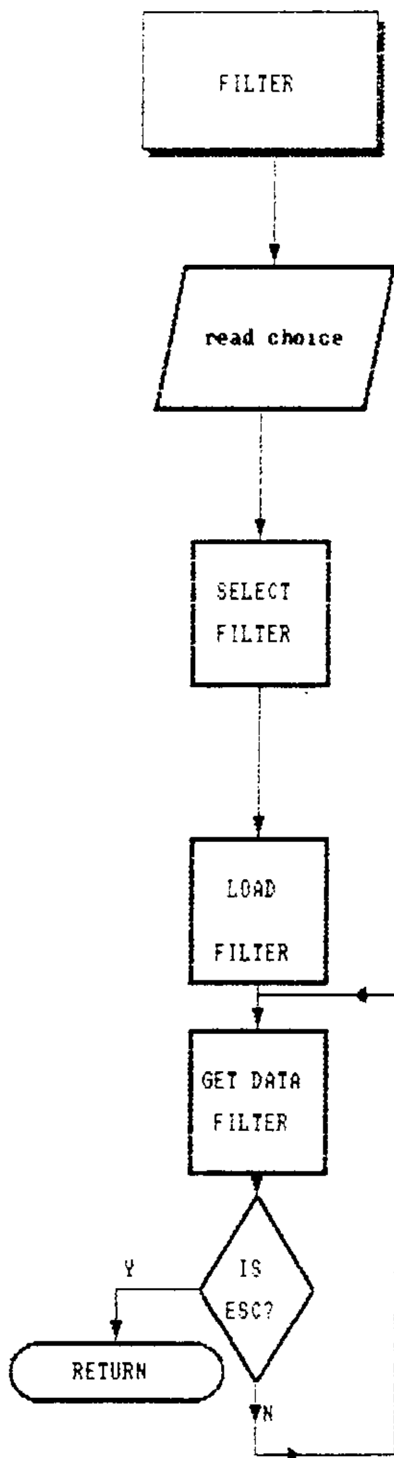
accumulator immediate (LACK), and load auxiliary register immediate (LARK).

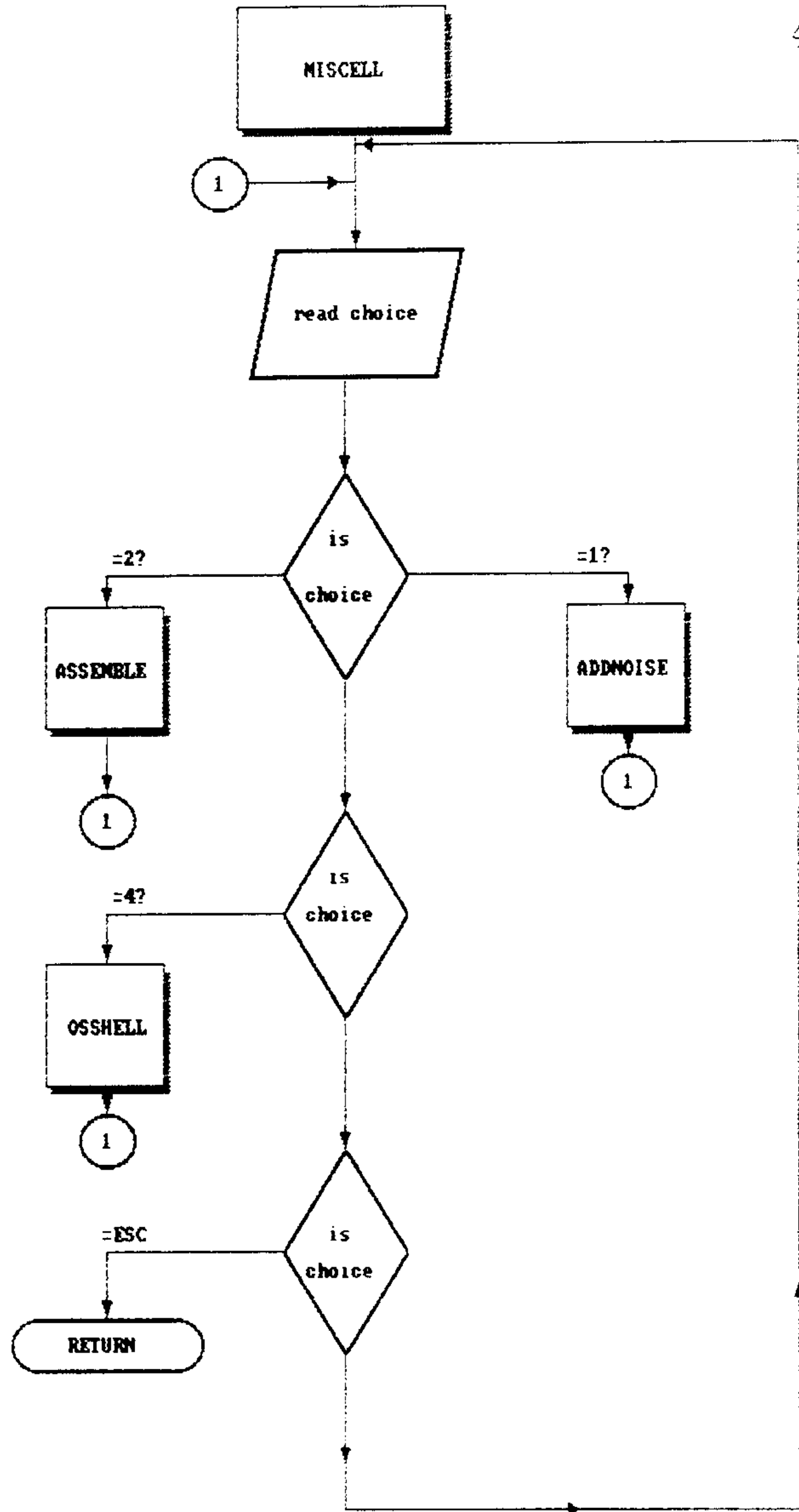
The following tables include the symbols and abbrevatrions that are used in the instruction set summary and in the instruction descriptions, the complete instruction set summary, and a description of each instruction.

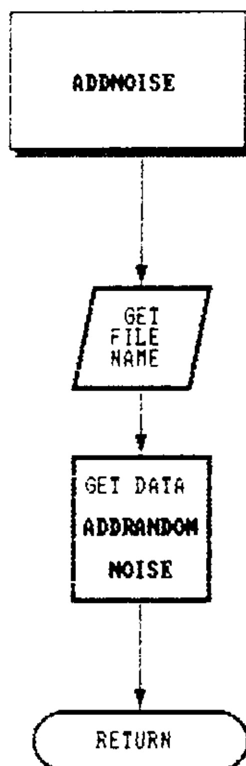
SYSTEM FLOW CHART




CAPTURE
read choice
IS
CHOICE
=2?
=1?
OFF-LINE
ON-LINE
GET
FILE
NAME
GET
FILE
NAME
GET DATA
AND
STORE
GET DATA
DISPLAY
STORE
RETURN


FILTER
read choice
SELECT
FILTER
LOAD
FILTER
GET DATA
FILTER
IS
ESC?
Y
N
RETURN


MISCELL
1
read choice
is
choice
=2?
=1?
ASSEMBLE
ADDNOISE
1
1
is
choice
=4?
OSSHELL
1
is
choice
=ESC
RETURN


ADDNOISE
GET
FILE
NAME
GET DATA
ADDRANDOM
NOISE
RETURN

```
#include<dos.h>
#include<dir.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>
#include<stdio.h>
#include<string.h>
#include<graphics.h>
#define CNTL            0x307
#define CNT2            0x306
#define STLEN              40
#define DBKCOL              0
#define HLP_CENTREX       325
#define HLP_CENTREY       235
#define HLP_VERT          176
#define HLP_HORIZ         352
#define HLP_BKCOL           3
#define HLP_BORD            4
#define MESG_CENTREX      325
#define MESG_CENTREY      300
#define MESG_BKCOL           4
#define MESG_BORD           15
#define MESG_TEXT_COL      0
#define ENTER             7181
#define ESC                283
```

```
#define F1              15104
#define F2              15360
#define UP              18432
#define DOWN            20480
#define RIGHT           19712
#define LEFT            19200
#define HI_FILL            12
#define HI_TEXT            10
#define HI_BORD            14
#define BK_FILL             3
#define BK_TEXT             4
#define BK_BORD            10
#define PULL_BOX_COL        8
#define PULL_FILL_COL       1
#define MAIN_BOX_COL        7
#define MAIN_FILL_COL       4
#define DISP_BOX_BK_COL  1
#define SIGNAL_COL         15

char mes[5][4][9]=  "CAPTURE","MISCELL","DISPLAY","FILTER",
                          "ONLINE","OFFLINE","","",
              "ADDNOISE","LOADFILE","ASSEMBLE","OSSHELL",
                          "REALTIME","RECORDED","","",
                          "FI1","FI2","FI3",""
                                ;

char fil[3][13]= "r1.bin","ii1.bin","" ;

char  message[35];

char esh[30]= "xasm320" ,

char  car,file[13];
```

```
fpos_t point[50];
struct viewporttype viewsets;
unsigned hlp_flag=1,i,j,k,
          left,top,right,count,bottom,orig,buf[4096];
int del=700,func_pointer,menu_ptr,xpointer,ypointer;
int d_x=350,d_y=175,d_v=200,d_h=400,step=1;
int far *mem_pointer = (int far *)0xd0000400;
int far *pptr = (int far *)0xd0000000;
long number=0;
struct textsettingstype t;
FILE *hlpptr;
struct  a2d
           int ad  :12;
            bits;
```

```
*
       IDT   'IIR1'
Y01    EQU    0
Y11    EQU    1
Y21    EQU    2
B01    EQU    3
B11    EQU    4
B21    EQU    5
A11    EQU    6
A21    EQU    7
XN     EQU    8
XTEMP  EQU    9
YN     EQU   10
YTEMP  EQU   11
OFF    EQU   12
ADD    EQU   13
ONE    EQU   14
DST    EQU   15
*
       AORG   0
       B      START
*
       DATA 0
       DATA 0
       DATA 0
```

```
       DATA >3D5
       DATA >7AB
       DATA >3D5
       DATA >3A24
       DATA >B685
       DATA 0
       DATA 0
       DATA 0
       DATA 0
       DATA >800
       DATA >200
COEF   DATA >1
*
START  LDPK  0
       LACK  1
       SACL  ONE
       LACK  2
       SACL  DST
       LARK  0,0
       LARK  1,COEF-2
       LAC   DST,0
LOP1   MAR   *,0
       TBLR  *+,1
       ADD   ONE,0
       BANZ  LOP1
```

```
        LARK   AR0,11
        LARK   AR1,255
HERE     BIOZ   NEXT
         B      HERE
NEXT     IN     XN,03
        LAC    XN,0
        SUB    OFF,0
        SACL   XTEMP
*
        LT     XTEMP
        MPY    B01
        ZALH   Y21
        APAC
        SACH   Y01,1
        MPY    B11
        ZALH   Y11
        LTA    Y01
        MPY    A11
        ADD    Y01,15
        APAC
        SACH   Y21,0
        MPY    A21
        PAC
        LT     XTEMP
        MPY    B21
```

```
APAC
SACH  Y11,0
LAC   Y01,0
ADD   OFF,0
SACL  YTEMP
LARP  0
LAC   ADD,0
TBLW  *,1
ADD   ONE,0
SACL  ADD,0
BANZ  HERE
END
```

```
*       IDT   'IIR2'
Y01   EQU    0
Y11   EQU    1
Y21   EQU    2
B01   EQU    3
B11   EQU    4
B21   EQU    5
A11   EQU    6
A21   EQU    7
XN    EQU    8
XTEMP EQU    9
YN    EQU   10
YTEMP EQU   11
OFF   EQU   12
ADD   EQU   13
ONE   EQU   14
DST   EQU   15
COUNT  EQU 16
*
      AORG  0
      B     START
*
      DATA 0
      DATA 0
      DATA 0
```

```
        DATA >8A2
        DATA >1144
        DATA >8A2
        DATA >124D
        DATA >CB2A
        DATA 0
        DATA 0
        DATA 0
        DATA 0
        DATA >800
        DATA >200
        DATA 3
COEF    DATA >1
*
START   LDPK   0
        LACK   1
        SACL   ONE
        LACK   2
        SACL   DST
        LARK   0,0
        LARK   1,COEF-2
        LAC    DST,0
LOP1    MAR    *,0
        TBLR   *+,1
        ADD    ONE,0
```

```
        BANZ   LOP1
        LARK   AR0,11
LOOP    LARK   AR1,255
HERE    BIOZ   NEXT
        B      HERE
NEXT    IN     XN,03
        LAC    XN,0
        SUB    OFF,0
        SACL   XTEMP
*
        LT     XTEMP
        MPY    B01
        ZALH   Y21
        APAC
        SACH   Y01,1
        MPY    B11
        ZALH   Y11
        LTA    Y01
        MPY    A11
        ADD    Y01,15
        APAC
        SACH   Y21,0
        MPY    A21
        PAC
        LT     XTEMP
```

```
MPY   B21
APAC
SACH  Y11,0
LAC   Y01,0
ADD   OFF,0
SACL  YTEMP
LARP  0
LAC   ADD,0
TBLW  *,1
ADD   ONE,0
SACL  ADD,0
BANZ  HERE
LAC   COUNT,0
SUB   ONE,0
SACL  COUNT
BNZ   LOOP
END
```

```
* IDT  'IIR3'
Y01   EQU   0
Y11   EQU   1
Y21   EQU   2
B01   EQU   3
B11   EQU   4
B21   EQU   5
A11   EQU   6
A21   EQU   7
XN    EQU   8
XTEMP EQU   9
YN    EQU   10
YTEMP EQU   11
OFF   EQU   12
ADD   EQU   13
ONE   EQU   14
DST   EQU   15
COUNT EQU   16
*
      AORG  0
      B     START
*
      DATA 0
      DATA 0
      DATA 0
      DATA >51C8
      DATA >DC6F
```

```
        DATA >51C8
        DATA >124D
        DATA >CB2A
        DATA 0
        DATA 0
        DATA 0
        DATA 0
        DATA >800
        DATA >200
        DATA 3
COEF    DATA >1
*
START   LDPK  0
        LACK  1
        SACL  ONE
        LACK  2
        SACL  DST
        LARK  0,0
        LARK  1,COEF-2
        LAC   DST,0
LOP1    MAR   *,0
        TBLR  *+,1
        ADD   ONE,0
        BANZ  LOP1
        LARK  AR0,11
LOOP    LARK  AR1,255
HERE    BIOZ  NEXT
        B     HERE
```

```
NEXT   IN    XN,03
       LAC   XN,0
       SUB   OFF,0
       SACL  XTEMP
*
       LT    XTEMP
       MPY   B01
       ZALH  Y21
       APAC
       SACH  Y01,1
       MPY   B11
       ZALH  Y11
       LTA   Y01
       MPY   A11
       ADD   Y01,15
       APAC
       SACH  Y21,0
       MPY   A21
       PAC
       LT    XTEMP
       MPY   B21
       APAC
       SACH  Y11,0
       LAC   Y01,0
       ADD   OFF,0
       SACL  YTEMP
       LARP  0
```

```
LAC    ADD,0
TBLW   *,1
ADD    ONE,0
SACL   ADD,0
BANZ   HERE
LAC    COUNT,0
SUB    ONE,0
SACL   COUNT
BNZ    LOOP
END
```

```
                IDT         'ADC1024'

                AORG        0
                B           START
*
COUNT           EQU         143
COUNT1          EQU         255
COUNT2          EQU         6
ZERO            EQU         0
ONE             EQU         1
TWO             EQU         2
ADPORT          EQU         3
STPORT          EQU         7
ROT             EQU         8
*
                AORG        >10
*
START           ROVM
                DINT
                LDPK        0
                ZAC
                LARK        0,COUNT
                LARP        0
LOOP            SACL        * -
                BANZ        LOOP
*
                OUT         0,STPORT
*
                LACK        ONE
                SACL        ONE
                ADD         ONE
                SACL        TWO
*
                LAC         TWO,ROT
                LARK        1,COUNT2
NLOOP           LARK        0,COUNT1
                LARP        0
                EINT
*
WAIT            BIOZ        ADC
                B           WAIT
*
*
ADC             IN          ZERO,ADPORT
                TBLW        ZERO
                ADD         ONE
                BANZ        WAIT
                LARP        1
                BANZ        NLOOP
*
OVER            OUT         ONE,STPORT
*
HERE            B           HERE
*
```

```
#include"dsp.h"

/*  function to set the sample rate of the ADC of VDSP100
    input : sample rate in KHZ
*/

sample_rate(long st)

        {

         int count_l,count_h;


          count=6250/st;

          count_h=(count>>8)&0x00ff;

          count_l=0x00ff&count;

          outport(CNTL,0x00b6);

          outport(CNT2,count_l);

          outport(CNT2,count_h);

        }


 /*

 function to explode box

 inputs: centre x_coordinate,y_coordinate,horiz_length,
 vert_lengthvalues of the box,background colour,border color.
 caution: graphics screen should be set before using this procedur
 */

 explode_box(int  centre_x,int  centre_y,int   vlength,int
 hlength,int bk_col,int bor_col)

       {

        int bcount=0,ratio;

        top=centre_y;

        left=centre_x;
```

```
right=centre_x;
bottom=centre_y;
vlength=vlength/2;
hlength=hlength/2;
ratio=ceil((double)hlength/vlength);
orig=getcolor();
setcolor(bk_col);
while(top>centre_y-vlength||left>centre_x-hlength)
{
bcount++;
if(left>centre_x-hlength)
{
left--;
right++;
}
if(bcount==ratio)
{
bcount=0;
if(top>centre_y-vlength)
{
top--;
bottom++;
}
}
rectangle(left,top,right,bottom);
```

```
 }

 setcolor(bor_col);

 left+=4;

 top+=4;

 right-=4;

 bottom-=4;

 rectangle(left--,top--,right++,bottom++);

 left-=2;

 top-=2;

 right+=2;

 bottom+=2;

 rectangle(left--,top--,right++,bottom++);

 setcolor(orig);
}

/*

    function to shrink box
    inputs: centre x_coordinate,y_coordinate values of the box,
    caution: graphics screen should be set before using this
    procedure

*/

 shrink_box(int  centre_x, int centre_y,int  vlength,int
 hlength,int bk_col)

 {

  int bcount=0,ratio;

  orig=getcolor();

  setcolor(bk_col);

  vlength=vlength/2;
```

```
hlength=hlength/2;
ratio=hlength/vlength;
top=centre_y-vlength;
bottom=centre_y+vlength;
left=centre_x-hlength;
right=centre_x+hlength;

while(top<centre_y||left<centre_x)
{
rectangle(left,top,right,bottom);
bcount++;
if(left<centre_x)
{
left++;
right--;
}
if(bcount==ratio)
{
bcount=0;
if(top<centre_y)
{
top++;
bottom--;
}
}
```

```
        }

        putpixel(centre_x,centre_y,DBKCOL);

        setcolor(orig);


       }

   /*

       function to  initialie graphics screen

   */

       init()

      {

        int gd=DETECT, gm;

        initgraph(&gd,&gm,"D:\\TC");

      }


/*
    function to explode box  with a message
    inputs: centre x_coordinate,y_coordinate,
    values of the box background colour,border color,
    pointer to the message.

    caution: graphics screen should be set before using this
    procedure
*/

explode_box_mes(int  centre_x,int  centre_y,int   bk_col,int
bor_col,int text_col)

     {

      int text_count,bcount=0,message_length,ratio,
top=centre_y,left=centre_x,right=centre_x,bottom=centre_y;

      int vlength,hlength,mes_v_length,mes_h_length,i,j=0;

      char s[2];


      s[1]='\0';
```

```
message_length=strlen(message);
hlength=message_length;
if(message_length>STLEN)
hlength=STLEN;
vlength=1;
if(message_length>STLEN)
vlength=message_length/STLEN+1;
mes_v_length=vlength;
mes_h_length=hlength;
vlength=(1+vlength)*10/2;
hlength=(2+hlength)*13/2;
ratio=hlength/vlength;
orig=getcolor();
setcolor(bk_col);
while(top>centre_y-vlength||left>centre_x-hlength)
{
bcount++;
if(left>centre_x-hlength)
{
left--;
right++;
}
if(bcount==ratio)
{
bcount=0;
```

```
if(top>centre_y-vlength)
{
top--;
bottom++;
}
}
rectangle(left,top,right,bottom);
}
setcolor(bor_col);
rectangle(left--,top--,right++,bottom++);
setcolor(text_col);
moveto(left+17,top+7);
for(text_count=0;text_count<mes_v_length;text_count++)
{
 for(i=0;i<mes_h_length;i++)
 {
  s[0]=message[j++];
  outtext(s);
  moverel(5,0);
  if(j>=message_length)
  break;
 }
 moveto(left+15,top+5+(1+text_count)*10);
}
setcolor(orig);
```

```
      }

/*
    function to shrink box  with a message
    inputs: centre x_coordinate,y_coordinate,horiz_length,
    values of the box background colour,pointer to the message.

    caution: graphics screen should be set before using this
    procedure
*/

shrink_box_mes(int centre_x,int centre_y,int bk_col)

      {

       int bcount=0,message_length,ratio,
top=centre_y,left=centre_x,right=centre_x,bottom=centre_y;

       int vlength,hlength;

       message_length=strlen(message);

       hlength=message_length;

       if(message_length>STLEN)

       hlength=STLEN;

       vlength=1;

       if(message_length>STLEN)

       vlength=message_length/STLEN+1;

       vlength=(1+vlength)*10/2;

       hlength=(2+hlength)*13/2;

       ratio=hlength/vlength;

       top=centre_y-vlength;

       bottom=centre_y+vlength;

       left=centre_x-hlength;

       right=centre_x+hlength;
```

```
orig=getcolor();
setcolor(bk_col);
while(top<centre_y||left<centre_x)
{
rectangle(left,top,right,bottom);
bcount++;
if(left<centre_x)
{
left++;
right--;
}
if(bcount==ratio)
{
bcount=0;
if(top<centre_y)
{
top++;
bottom--;
}
}

}
setcolor(orig);
}
/*
function to read integers in graphics mode.
```

```
        input :   size of the integer to be read,x,y position.
      caution : this function needs graphics initialization.
*/
   long read_integer(int size,int x,int y)

        {

        char s[5];

        number=0;

        count=0;

        gettextsettings(&t);

        s[1]='\0';

        orig=getpixel(x-(size/2*5),y-5);

        setcolor(5);

        settextstyle(DEFAULT_FONT,HORIZ_DIR,1);

        rectangle(x-(size/2*5),y-5,x+(size/2*25),y+10);

        moveto(x,y);

        while(count<size)

        {

         car=getch();

         if(isdigit(car))

         {

         s[count]=car;

         s[count+1]='\0';

         outtext(s+count);

         number=number*10+car-48;

         count++;

         }

         else
```

```
         if(car=='\r')
         break;
        }
        setfillstyle(SOLID_FILL,orig);
        setcolor(orig);
        moveto(x,y);
        outtext(s);
        floodfill(x,y,5);
        settextstyle(t.font,t.direction,t.charsize);
        rectangle(x-(size/2*5),y-5,x+(size/2*25),y+10);
        setcolor(orig);
        return(number);

        }
/*

Function to load help array with the corresponding page numbers
         the array to be loaded is declared globally.
caution : this function needs the help file hlp.dat to load
                  the page nos.

*/
load_helparray()

      {

        fpos_t temp=0;
        char c;

        i=1;
        point[0]=0;
        if((hlpptr=fopen("hlp.dat","r"))==NULL)
```

```
        {
      strcpy(message,"HLP.DAT FILE NOT FOUND");
      explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
      hlp_flag=0;
      getch();
      shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);
          return(1);
        }
      fsetpos(hlpptr,0);
      while((c=fgetc(hlpptr))!='`')
        {
          temp++;
          fsetpos(hlpptr,&temp);
          if(c=='~')
            {
             point[i]=temp;
             i++;
            }

         }
         return(0);
}
```

/*
This function displays the page specified in a window it requires a key press or amouse click to come out of the window

caution: This function assumes that the global array

```
     for help page pointes are allready loaded.

*/
hlp_disp(unsigned page)

      {

         long line_step;

         char c,t[2];


            if(hlp_flag==0) return;

            explode_box(HLP_CENTREX,HLP_CENTREY,
HLP_VERT,HLP_HORIZ,HLP_BKCOL,HLP_BORD);

            strcpy(message,"HELP");

            explode_box_mes(HLP_CENTREX,top+10,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);

            moveto(left+7,top+23);

            settextstyle(DEFAULT_FONT,0,1);

            fsetpos(hlpptr,&point[page-1]);

            line_step=0;

            t[1]='\0';

            while((c=fgetc(hlpptr))!='~')

             {

              if(c=='\n')

              {

               line_step+=12;

               moveto(left+7,top+23+line_step);

              }

              else

               {
```

```
                if(c=='\t')
                {
                for(i=0;i<4;i++)
                {
                t[0]=' ';
                outtext(t);
                }
                }
                else
                {
                t[0]=c;
                setcolor(1);
                outtext(t);
                setcolor(0);
                }
                }
              }
            bioskey(0);
           shrink_box(HLP_CENTREX,HLP_CENTREY,
HLP_VERT,HLP_HORIZ,DBKCOL);
  }
  /*
    function to load a binary file onto vdsp-100,
    starting from the address d000:0000.
  input: pointer to the filename that is to be loaded to dsp
  */
```

```
load(data)

char *data;

{

FILE *stream;

count=inp(0x307);

stream=fopen(data,"rb");

if(stream==NULL)

{

   strcpy(message,"FILE NOT FOUND");

   explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);

   getch();

   shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);

   return(1);

}

fread(buf,2,4096,stream);

for(count=0;count<4096;count++)

 {

   *(pptr+count)=buf[count];

 }

 fclose(stream);

 }

 /*function to load a binary file from vdsp-100,
 starting from the address d000:0000.
 the file name is given by the string data*/


 sdata(data)
```

```
char *data;
{
FILE *stream;

i=inport(0x307);
stream=fopen(data,"wb");
if(stream==NULL)
{
   strcpy(message,"FILE NOT FOUND");
   explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
   getch();
   shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);
   return(1);
}
 for(i=0;i<4096;i++)
 {
   buf[i]= *(pptr+i);
 }
 fwrite(buf,2,4096,stream);
 fclose(stream);
 }


disp_real()
     {
```

```
        explode_box(d_x,d_y,d_v+5,d_h+5,DISP_BOX_BK_COL,4);
        while(1)
        {
        step=1;
        load("adc.bin");
        strcpy(message,"SAMPLING RATE");
        explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
        i=read_integer(3,440,300);
        if(i==0)i=100;
        shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);
        sample_rate(i);
        while(((k=bioskey(1))==0)||(k==F2))
        {
        i=inp(0x306);
        delay(150);
        i=inp(0x307);
        for(count=0;count<2048;count++)
          {
           buf[count]=*(mem_pointer+count);
          }
          win_sig();
         }
k=bioskey(0);
switch(k)
```

```
    {
    case        F1:  shrink_box(d_x,d_y,d_v+5,d_h+5,0);
                     hlp_disp(21);
              explode_box(d_x,d_y,d_v+5,d_h+5,DISP_BOX_BK_COL,4);
                     break;

    case        ESC:shrink_box(d_x,d_y,d_v+5,d_h+5,0);
                    return(0);
  }
}
}


add_noise_file(char source[])
     {
      FILE *stream;
      stream=fopen(source,"rb+wb");
      if(stream==NULL)
{
   strcpy(message,"FILE NOT FOUND");
   explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
   getch();
   shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);
   return(1);
}
      explode_box(d_x,d_y,d_v+5,d_h+5,DISP_BOX_BK_COL,4);
```

```
while(1)
{
i= fread(buf,2,1024,stream);
if(i!=1024)
break;
for(count=0;count<1024;count++)
{
    buf[count]+=random(500);
}
win_sig();
k= fwrite(buf,2,1024,stream);
}
shrink_box(d_x,d_y,d_v+5,d_h+5,0);
fclose(stream);
}
disp_file(char filename[])
{
 size_t bytecount;
 FILE *stream;

 stream=fopen(filename,"rb");
 if(stream==NULL)
{
  strcpy(message,"FILE NOT FOUND");
  explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
```

```
getch();
shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);
return(1);
}
explode_box(d_x,d_y,d_v+5,d_h+5,DISP_BOX_BK_COL,4);
while(eof(stream))
{
bytecount=fread(buf,2,1024,stream);
if(bytecount!=1024)
break;

win_sig();
}
shrink_box(d_x,d_y,d_v+5,d_h+5,0);
fclose(stream);
}


capture_file(int showflag)
{
int page;
struct ffblk  ff;
FILE *stream;


read_string(file,13);
strcpy(message,"NO OF PAGES PLEASE");
explode_box_mes(250,175,MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
```

```
page=read_integer(2,400,175);
shrink_box_mes(250,175,DBKCOL);
if(page==0)return(2);
number=findfirst(file,&ff,FA_ARCH);
if(number==0)
        {
         strcpy(message,"FILE ALREADY EXISTS!!");
         explode_box_mes(250,175,MESG_BKCOL,
MESG_BORD,MESG_TEXT_COL);
         strcpy(message,"PRESS ESC TO CANCEL");
         explode_box_mes(450,295,MESG_BKCOL,
MESG_BORD,MESG_TEXT_COL);
         while((number=bioskey(1))==0);
         if(number!=0)
         bioskey(0);
         if(number==ESC)
         {
         shrink_box_mes(450,295,DBKCOL);
         strcpy(message,"FILE ALREADY EXISTS!!");
         shrink_box_mes(250,175,DBKCOL);
            return(3);
        }
         shrink_box_mes(450,295,DBKCOL);
         strcpy(message,"FILE ALREADY EXISTS!!");
         shrink_box_mes(250,175,DBKCOL);
         }
```

```
stream=fopen(file,"wb");
if(stream==NULL)
   {
    strcpy(message,"FILE CREATION ERROR");
    explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
    getch();
    shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);
    return(1);
   }
  load("adc.bin");
  strcpy(message,"SAMPLING RATE");
  explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
  i=read_integer(3,440,300);
  if(i==0) i=100;
  shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);
  sample_rate(i);
  if(!showflag)
    {
         strcpy(message,"CAPTURING");
         explode_box_mes(250,175,MESG_BKCOL,
MESG_BORD,MESG_TEXT_COL);
    }
   if(showflag)
   explode_box(d_x,d_y,d_v+5,d_h+5,DISP_BOX_BK_COL,4);
```

```
strcpy(message,"PAGE");
explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
for(k=0;k<page;k++)
{
  disp_integer(k+1,380,300,5);
  i=inp(0x306);
  delay(40);
  i=inp(0x307);
  for(count=0;count<=1024;count++)
    {
     buf[count]=*(mem_pointer+count);
    }
  if(showflag)
  win_sig();
  fwrite(buf,2,1024,stream);
  disp_integer(k+1,380,300,0);
 }
shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);
   if(!showflag)
  { strcpy(message,"CAPTURING");
   shrink_box_mes(250,175,DBKCOL);}
if(showflag)
shrink_box(d_x,d_y,d_v+5,d_h+5,0);
fclose(stream);
}
```

```
read_string(char file[],int size)
 {
 int x=400,y=175;

count=0;
gettextsettings(&t);
file[1]='\0';
orig=getpixel(x-(size/2*5),y-5);
strcpy(message,"FILENAME PLEASE ");
explode_box_mes(250,175,MESG_BKCOL,
MESG_BORD,MESG_TEXT_COL);
setcolor(5);
settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
rectangle(x-(size/2*5),y-5,x+(size/2*25),y+10);

moveto(x,y);
while(count<size)
{
 car=getch();
 if((isascii(car))&&(car!='\b')&&(car!='\r'))
 {
 file[count]=car;
 file[count+1]='\0';
 moveto(x,y);
 outtext(file);
```

```
count++;
}
else
{
 if(car=='\r')
 break;
 if((car=='\b')&&(count>0))
 {
 setcolor(0);
 moveto(x,y);
 outtext(file);
 count--;
 file[count]='\0';
 setcolor(5);
 moveto(x,y);
 outtext(file);
 }
}
}
 setcolor(DBKCOL);
 moveto(x,y);
 outtext(file);
 rectangle(x-(size/2*5),y-5,x+(size/2*25),y+10);
 shrink_box_mes(250,175,DBKCOL);
 setcolor(orig);
```

```
return;

}
win_sig()
{
    float i;
    if((number=bioskey(1))==F2)
    {
     strcpy(message,"STEP VALUE");
     explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
     step=read_integer(3,440,300);
     if(step<=0)
     step=1;
     shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);
    }
    j=d_x-d_h/2+5;
    for(count=20;j<=d_x+d_h/2-35;count++)
    {
    j+=step;
    bits.ad=buf[count];
      i=bits.ad+2048;
    i=((i*d_h)/4096)-200;
    putpixel(j,(int)(d_y+d_v/2-i),SIGNAL_COL);
    }

    j=d_x-d_h/2+5;
```

```
            if(step>5)
            delay(50);
             for(count=20;j<=d_h+d_x/2-35;count++)
            {
            j+=step;
            bits.ad=buf[count];
            i=bits.ad+2048;
            i=((i*d_h)/4096)-200;

            putpixel(j,(int)(d_y+d_v/2-i),DISP_BOX_BK_COL);
            }
}
iir(char filename[])
      {
        load(filename);
        sample_rate(100);
        explode_box(d_x,d_y,d_v+5,d_h+5,DISP_BOX_BK_COL,4);
        while(((i=bioskey(1))==0)||(i==F2))
        {
        i=inp(0x306);
        delay(100);
        i=inp(0x307);

       for(count=0;count<1024;count++)
           {
            buf[count]=*(mem_pointer+count);
```

```
        }
      win_sig();
      }
      shrink_box(d_x,d_y,d_v+5,d_h+5,0);
    }
/*
    function to display integers in graphics mode.

      input :   size of the integer to be read,x,y position.
    caution : this function needs graphics initialization.
*/
   disp_integer(int size,int x,int y,int col)
       {
       char s[5];
       count=0;
       s[0]='0';
       orig=getcolor();
       setcolor(col);
       moveto(x,y);
       while(size!=0)
       {
        car=size%10;
        size /=10;
        s[count++]=car+48;
        }
        s[count]='\0';
        strrev(s);
        outtext(s);
        return(1);
        }
```

```
#include"hlp1.c"

          main_hilt(int xpointer,int ypointer,int
               fill_col,int tex_col,int border,char str[])

                {
                   orig=getcolor();

                   setcolor(border);

                   rectangle(xpointer,ypointer,
xpointer+8*11,ypointer+15);

                   setcolor(fill_col);

                   outtextxy(xpointer+10,ypointer+5,str);

                   setfillstyle(SOLID_FILL,fill_col);

                   floodfill(xpointer+1,ypointer+1,border);

                   setcolor(tex_col);

                   outtextxy(xpointer+10,ypointer+5,str);

                   setcolor(orig);

                 }


          pull(int menu_ptr)

             {

                   int loc_menu_ptr,len,xpointer,ypointer;


                   len=3;

                   if(menu_ptr==0)

                    len=2;

                   if(menu_ptr==2)
```

```
len=2;

orig=getcolor();

setcolor(PULL_BOX_COL);

xpointer=menu_ptr*140+60-6;

ypointer=39;

rectangle(xpointer,ypointer,
xpointer+100,ypointer+len*20+6);

setfillstyle(SOLID_FILL,PULL_FILL_COL);

floodfill(xpointer+3,ypointer+3,PULL_BOX_COL);

xpointer=menu_ptr*140+60;

for(i=0;i<len;i++)

{

ypointer=i*20+45;

main_hilt(xpointer,
ypointer,BK_FILL,BK_TEXT,BK_BORD,mes[menu_ptr+1][i]);

}

loc_menu_ptr=0;

xpointer=menu_ptr*140+60;

ypointer=loc_menu_ptr*20+45;

main_hilt(xpointer,ypointer,
HI_FILL,HI_TEXT,HI_BORD,mes[menu_ptr+1][0]);

fflush(stdin);

i=bioskey(1);

while(i!=ESC)

{
```

```
                    switch(i)
                     {
                case ENTER :while((i=bioskey(1))!=0)bioskey(0);
                                   xpointer=menu_ptr*140+60-6;
                                   ypointer=39;
                                   setfillstyle(EMPTY_FILL,0);
                            floodfill(xpointer+3,ypointer+3,0);
                                   setcolor(0);
                            rectangle(xpointer,ypointer,
xpointer+100,ypointer+len*20+6);
                                   setcolor(orig);
                  return(1+(menu_ptr)*4+loc_menu_ptr);

                     case  DOWN :main_hilt(xpointer,
ypointer,BK_FILL,BK_TEXT,BK_BORD,mes[menu_ptr+1][loc_menu_ptr])
                                   if(loc_menu_ptr<len-1)
                                     loc_menu_ptr++;
                                   else
                                     loc_menu_ptr=0;
                                   xpointer=menu_ptr*140+60;
                                   ypointer=loc_menu_ptr*20+45;
                                   main_hilt(xpointer,ypointer,
HI_FILL,HI_TEXT,HI_BORD,mes[menu_ptr+1][loc_menu_ptr]);
                                   break;
                     case  UP  :main_hilt(xpointer,ypointer,
```

```
BK_FILL,BK_TEXT,BK_BORD,mes[menu_ptr+1][loc_menu_ptr]);
                                        if(loc_menu_ptr>0)
                                          loc_menu_ptr--;
                                        else
                                          loc_menu_ptr=len-1;
                                        xpointer=menu_ptr*140+60;
                                        ypointer=loc_menu_ptr*20+45;
                                        main_hilt(xpointer,ypointer,
HI_FILL,HI_TEXT,HI_BORD,mes[menu_ptr+1][loc_menu_ptr]);
                                        break;

                          case  F1 : bioskey(0);
                             hlp_disp(5+(menu_ptr)*4+loc_menu_ptr);

                          }
                       if(i!=0) bioskey(0);
                       i=bioskey(1);
                     }
                       while((i=bioskey(1))!=0)bioskey(0);
                       xpointer=menu_ptr*140+60-6;
                       ypointer=39;
                       setfillstyle(EMPTY_FILL,0);
                       floodfill(xpointer+3,ypointer+3,0);
                       setcolor(0);
                       rectangle(xpointer,ypointer,
xpointer+100,ypointer+len*20+6);
```

```
                    setcolor(orig);
                    return(0);
}
    main()
        {
                    init();
                    load_helparray();
                    orig=getcolor();
/*                  explode_box(325,175,300,600,1,3);*/
                    setcolor(MAIN_BOX_COL);
                    rectangle(55,10,570,35);
                    setfillstyle(SOLID_FILL,MAIN_FILL_COL);
                    floodfill(65,19,MAIN_BOX_COL);
                    for(i=0;i<4;i++)
                    {
                     xpointer=i*140+60;
                     ypointer=15;
                     main_hilt(xpointer,ypointer,
BK_FILL,BK_TEXT,BK_BORD,mes[0][i]);
                    }
                    fflush(stdin);
                    menu_ptr=0;
                    xpointer=menu_ptr*140+60;
```

```
                    ypointer=15;

                    main_hilt(xpointer,ypointer,
HI_FILL,HI_TEXT,HI_BORD,mes[0][menu_ptr]);

                    i=bioskey(1);

                    while(i!=ESC)

                    {

                     if(i!=0) bioskey(0);

                      switch(i)

                       {

                       case ENTER :  bioskey(0);

                                     func_pointer=pull(menu_ptr);

                                     if(func_pointer!=0)

                                     {

                                     call_func(func_pointer);

                                     }

                                     break;

                       case RIGHT :main_hilt(xpointer,
ypointer,BK_FILL,BK_TEXT,BK_BORD,mes[0][menu_ptr]);

                                   if(menu_ptr<3)

                                     menu_ptr++;

                                   else

                                     menu_ptr=0;

                                   xpointer=menu_ptr*140+60;

                                   ypointer=15;
```

```
                                      main_hilt(xpointer,ypointer,
HI_FILL,HI_TEXT,HI_BORD,mes[0][menu_ptr]);

                                      break;

                             case LEFT  :main_hilt(xpointer,ypointer,
BK_FILL,BK_TEXT,BK_BORD,mes[0][menu_ptr]);

                                      if(menu_ptr>0)

                                        menu_ptr--;

                                      else

                                        menu_ptr=3;

                                      xpointer=menu_ptr*140+60;

                                      ypointer=15;

                                      main_hilt(xpointer,ypointer,
HI_FILL,HI_TEXT,HI_BORD,mes[0][menu_ptr]);

                                      break;

                             case F1     : hlp_disp(menu_ptr+1);

                             }

                   i=bioskey(1);

                 }

 setcolor(orig);

 fclose(hlpptr);

}

call_func(int func_pointer)

  {
```

```
int size;
struct ffblk  ff;
char *err;
  switch(func_pointer)
     {
     case  1: number=3;
              while(number==3)
              number=capture_file(1);
              break;

     case  2: number=3;
              while(number==3)
              number=capture_file(0);
              break;
     case  5:read_string(file,13);
             add_noise_file(file);
             break;
     case  6:  setcolor(0);
               read_string(file,13);
               size=findfirst(file,&ff,FA_ARCH);
               if(size==-1)
               {
                err=strerror(errno);
                strcpy(message,err);
```

```
size=strlen(err);
message[size-1]='\0';
explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
getch();
else
{
size=findfirst("XASM320.EXE",&ff,FA_ARCH);
if(size==-1)
{
strcpy(message,"CAN'T FIND XASM320.EXE");
explode_box_mes(MESG_CENTREX,MESG_CENTREY,
MESG_BKCOL,MESG_BORD,MESG_TEXT_COL);
getch();
shrink_box_mes(MESG_CENTREX,MESG_CENTREY,DBKCOL);
}
else
{
strcpy(message,"ASSEMBLING");
explode_box_mes(250,175,MESG_BKCOL,
MESG_BORD,MESG_TEXT_COL);
strcat(esh," ");
strcat(esh,file);
strcat(esh,";  > err.dat");
system(esh);
```

```
                    shrink_box_mes(250,175,DBKCOL);
                    }
                    }
                    break;
        case  7:  closegraph();
                  system("c:\command.com");
                  init();
                  setcolor(MAIN_BOX_COL);
                  rectangle(55,10,570,35);
                  setfillstyle(SOLID_FILL,MAIN_FILL_COL);
                  floodfill(65,19,MAIN_BOX_COL);
                  for(i=0;i<4;i++)
                  {
                   xpointer=i*140+60;
                   ypointer=15;
                   main_hilt(xpointer,ypointer,BK_FILL,
BK_TEXT,BK_BORD,mes[0][i]);
                  }
                  fflush(stdin);
                  xpointer=menu_ptr*140+60;
                  ypointer=15;
                  main_hilt(xpointer,ypointer,HI_FILL,
HI_TEXT,HI_BORD,mes[0][menu_ptr]);
                 break;
        case  9:disp_real();
```

```
            break;
    case  10: setcolor(0);
              read_string(file,13);
              disp_file(file);
              break;

    case  13: step=5;
              iir(fil[0]);
              getch();
             break;
    case  14: step=5;
              iir(fil[1]);
              getch();
             break;

    }

}
```

CAPTURE | MISCELL | DISPLAY | FILTER

HELP

FILTERS

This option is for filtering in coming signals.

There are three filters in it.

They are explained in their respective help menu.

CAPTURE | MISCELL | DISPLAY | FILTER

ONLINE
OFFLINE

HELP

OFFLINE

This option is for capturing data online.
the data captured is displayed on the
monitor.
This captured data is stored in a file
specified by you.

IF the file already exists you are
cautioned.
You are also asked for the sample rate.

CAPTURE MISCELL DISPLAY FILTER

ADDNOISE
ASSEMBLE
OSSHELL

HELP

ASSEMBLE

This option assembles a DSP assembly program.

It requires XASM320.EXE file in the logged directory.

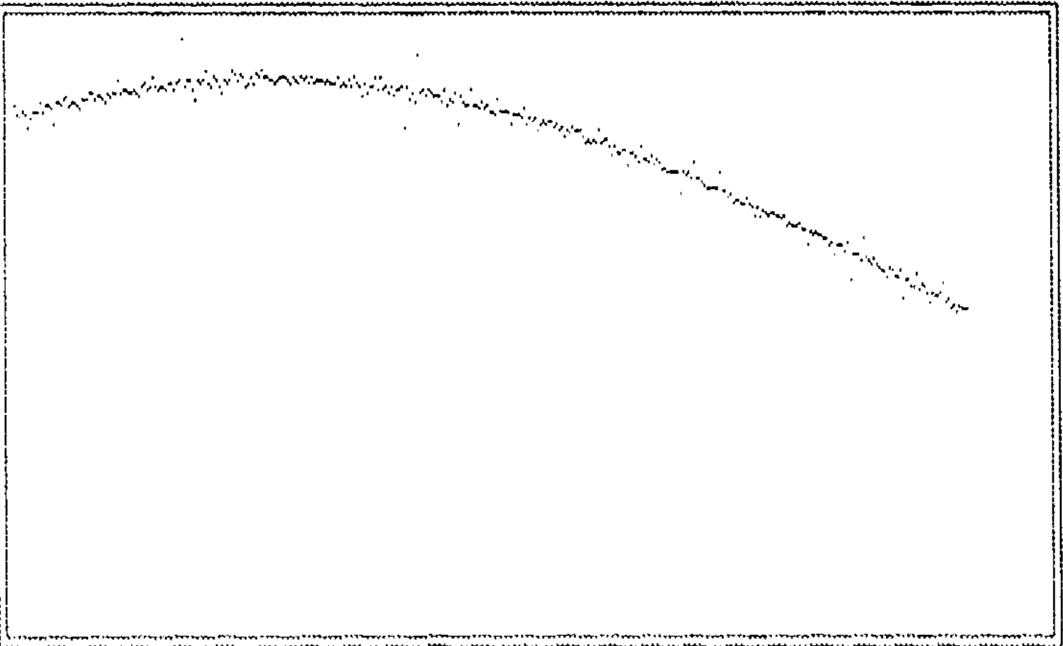
The error or warning messages if present are stored in file ERR.DAT.

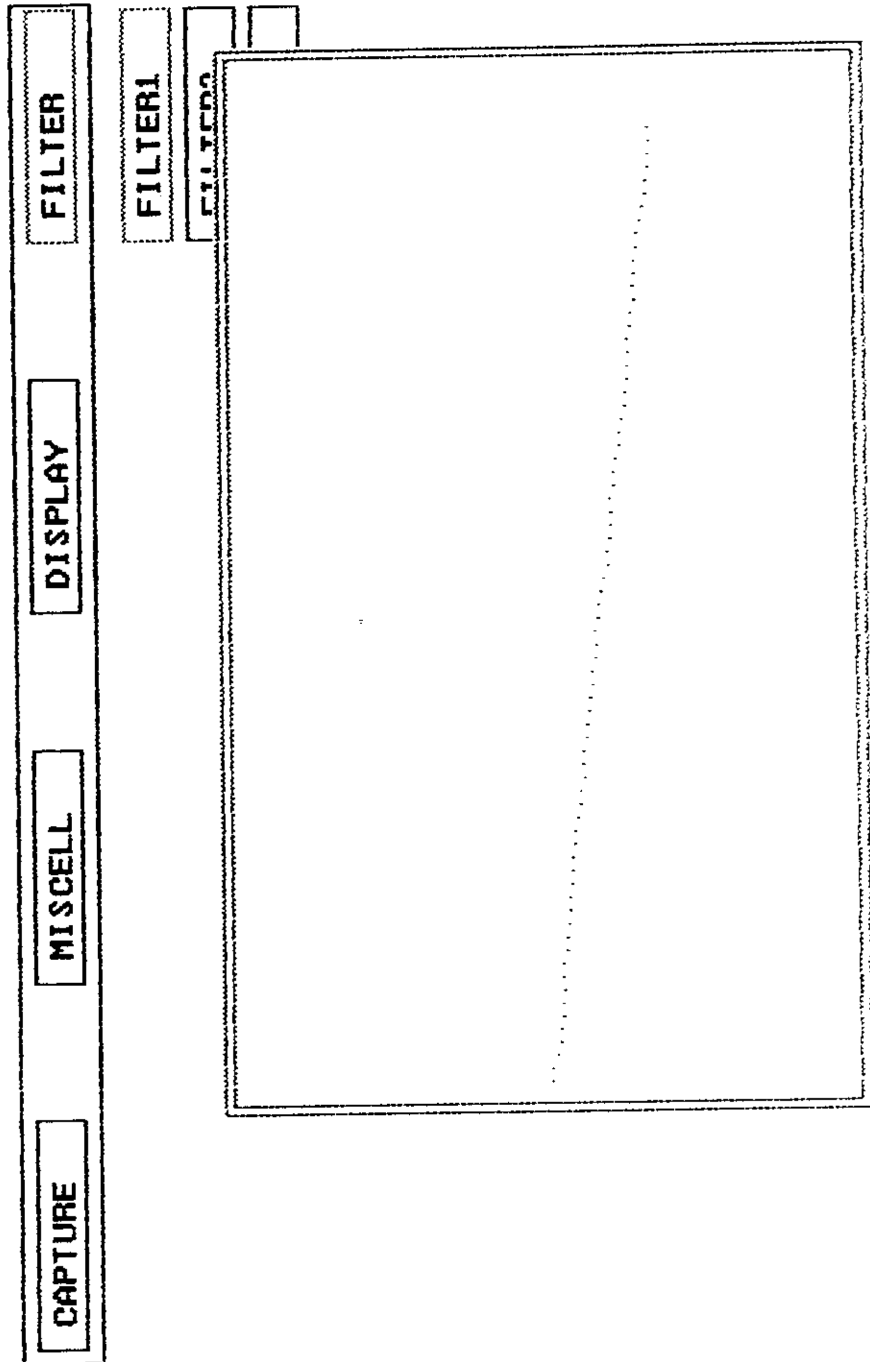
SIGNAL FREQUENCIES FOR THE OUPUTS

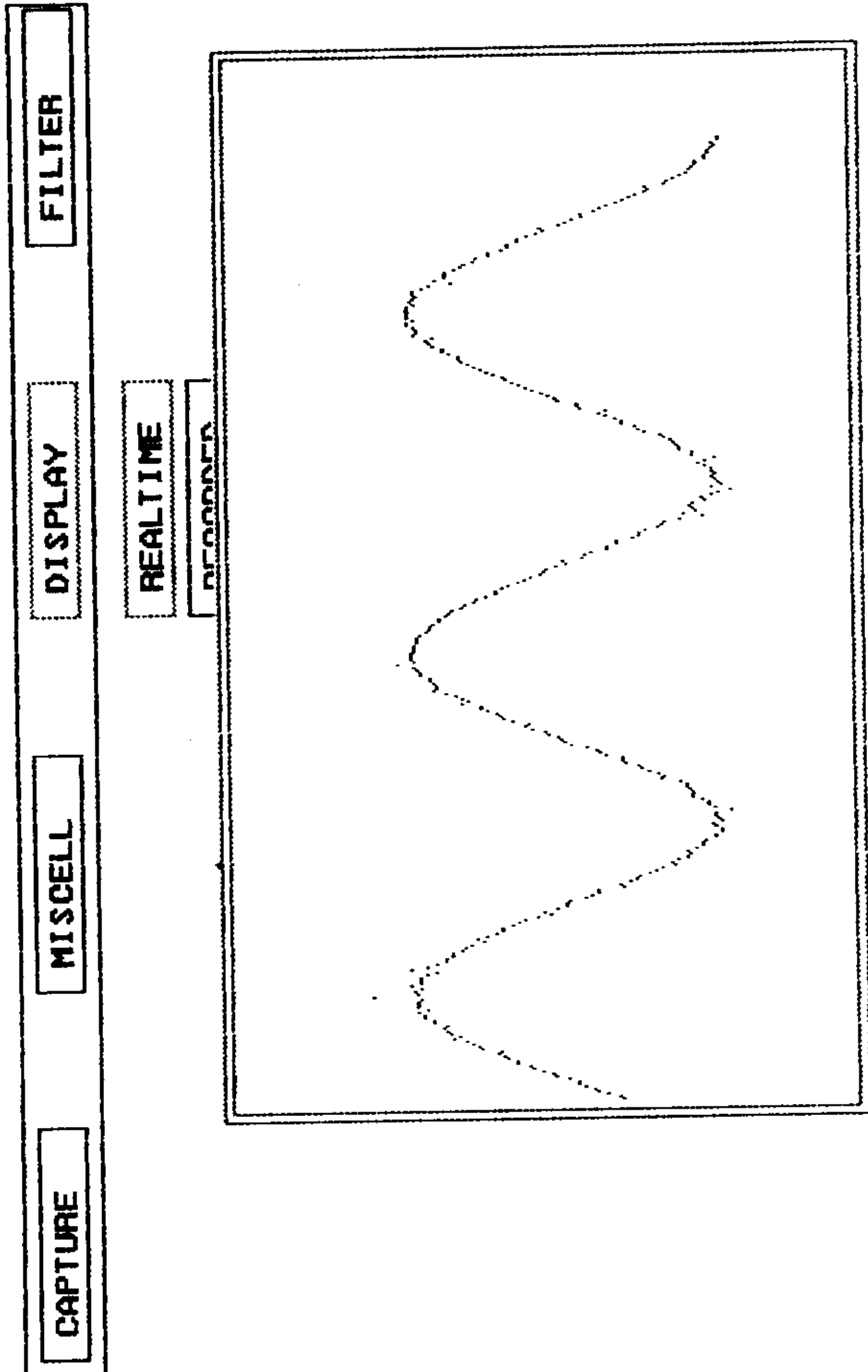
OUTPUT NO	FREQUENCY
1.	89 HZ
2.	89 HZ
3.	792 HZ
4.	792 HZ
5.	1.7 KHZ
6.	5.5 KHZ
7.	9.3 KHZ
8.	40.9 KHZ

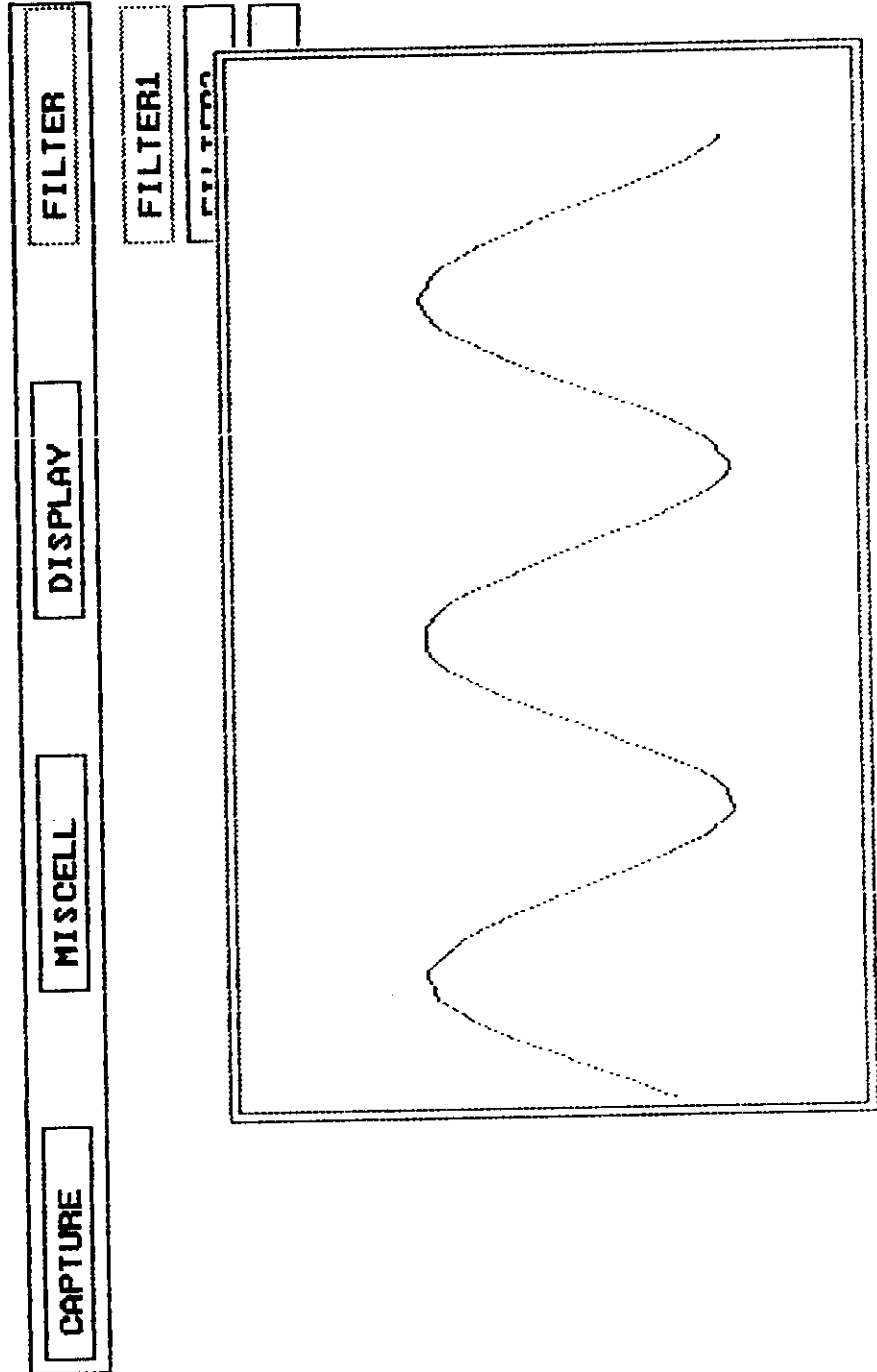
CAPTURE MISCELL DISPLAY FILTER

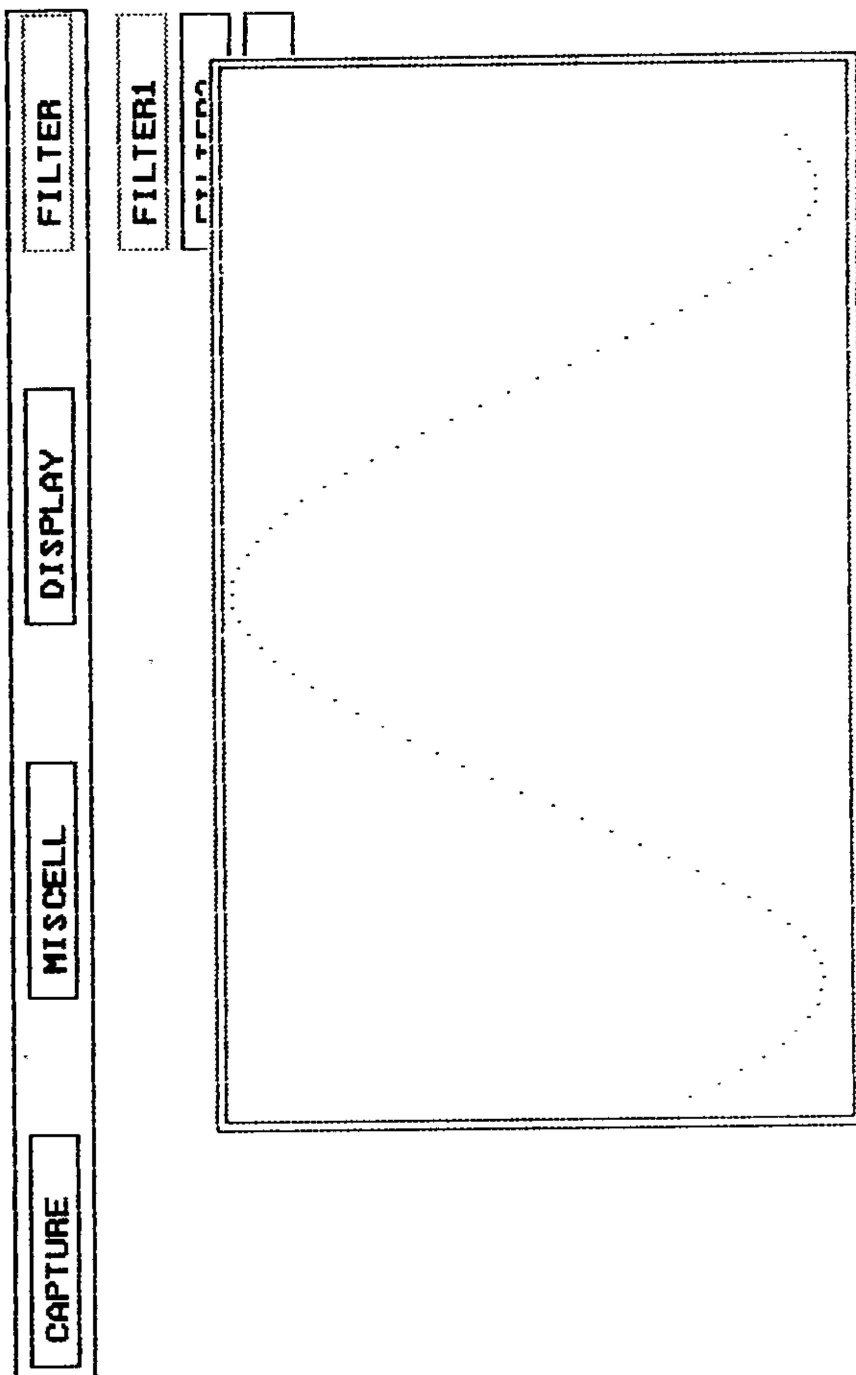
REALTIME

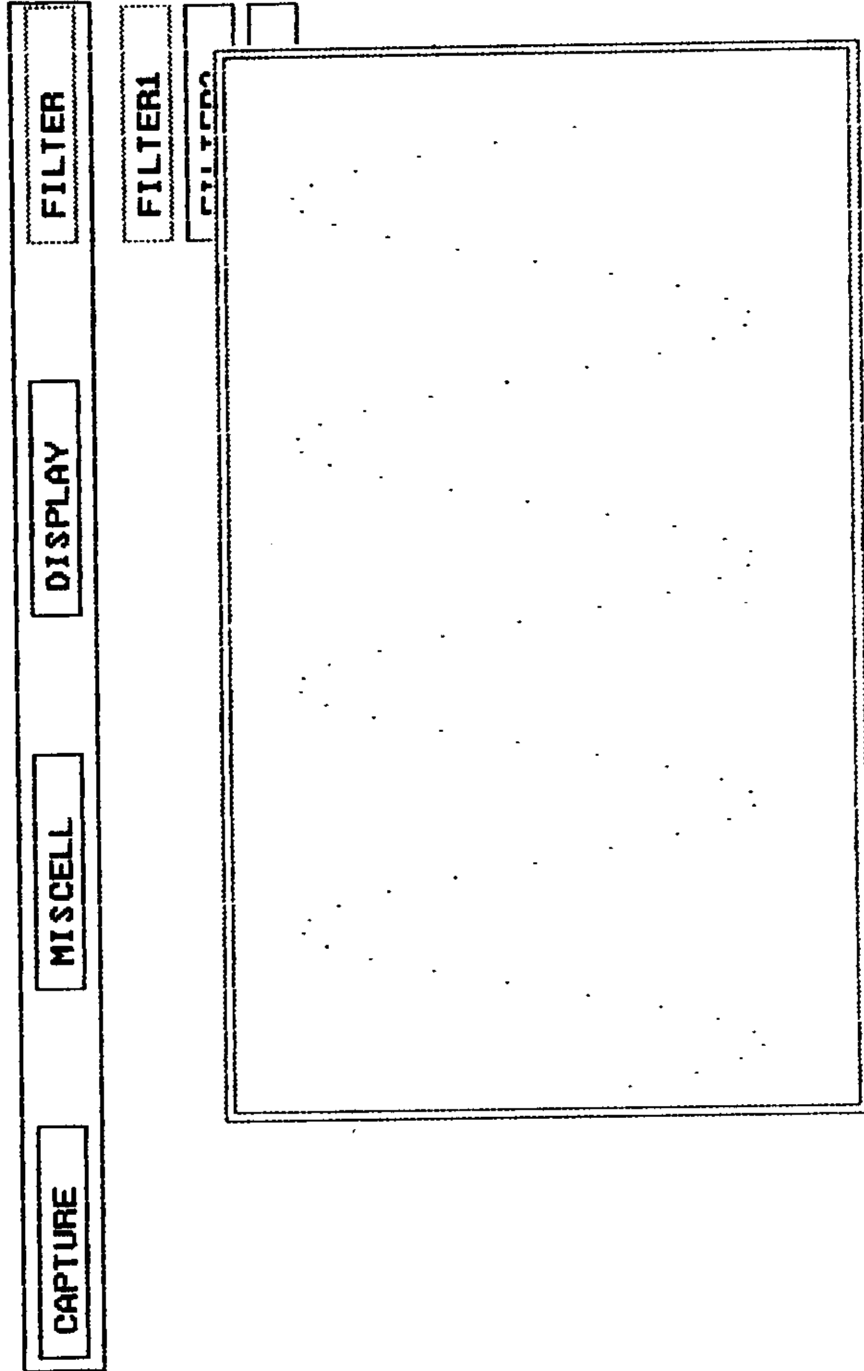


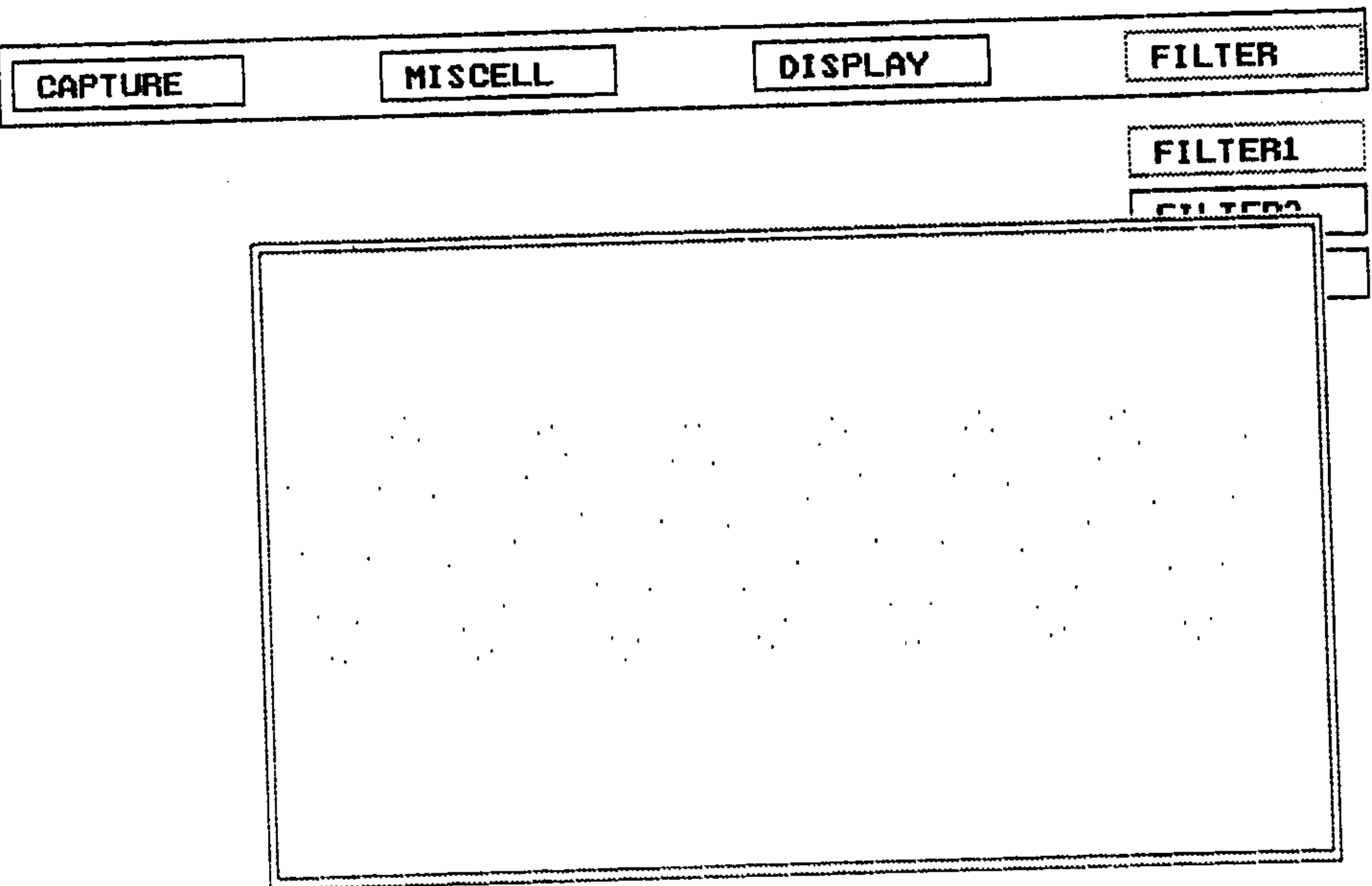

CAPTURE
MISCELL
DISPLAY
FILTER
FILTER1

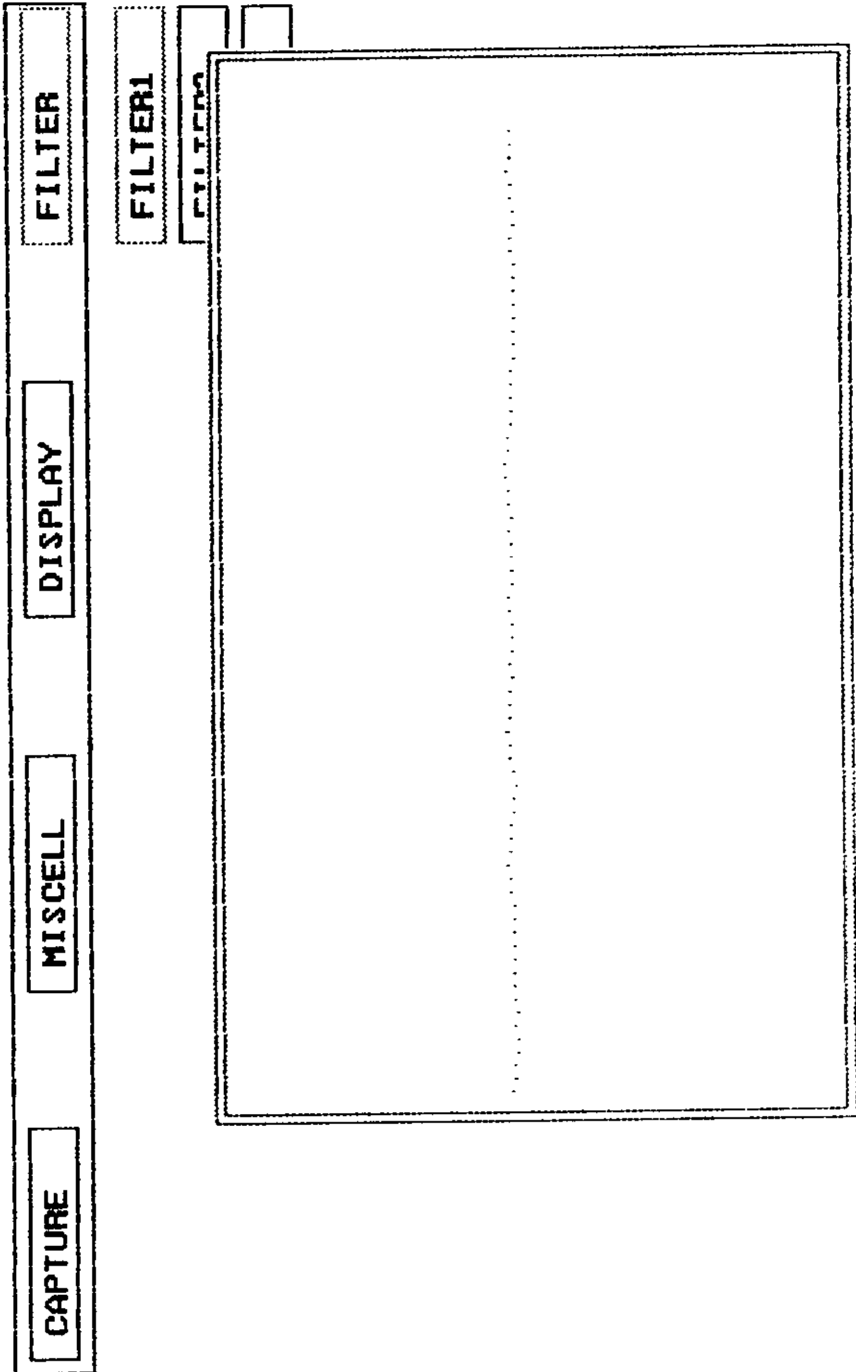

CAPTURE
MISCELL
DISPLAY
FILTER
REALTIME


FILTER
DISPLAY
MISCELL
CAPTURE
FILTER1


CAPTURE
MISCELL
DISPLAY
FILTER
FILTER1


CAPTURE
MISCELL
DISPLAY
FILTER
FILTER1


CAPTURE
MISCELL
DISPLAY
FILTER
FILTER1


CAPTURE
MISCELL
DISPLAY
FILTER
FILTER1

Conclusion

CONCLUSION

A software has been developed to implement three Butterworth type IIR filters,

1. A low pass filter with cut-off frequency 5 KHZ.
2. A low pass filter with cut-off frequency 10 KHZ.
3. A high pass filter with cut-off frequency 10 KHZ.

using TMS32010 assembly language. C language is used to integrate the chip operation under a pc's control.

Apart from this this software is added with facilities to display a real-time signal,to record a real-time signal in a file. the sampling rate for any signal processing operation is of user's choice. moreover facilities to add noise to a recorded file and to assemble a TMS32010 assembly file are provided.

This project is a novel attempt using the TMS32010 chip for DSP applications.This software can be used to demonstrate digital signal processing applications,can be used to demonstrate the advantage of TMS32010 chip in signal processing.

REFERENCES

1. Alan v. Oppenheim,Ronald W.Schafer , "Descrete-Time Signal processing",Printice-Hall,1989.

2. Roman Kuc, "Introduction to Digital Procesing", McGraw Hill, Singapore,1982.

3. John G.Proakis,Dimitris G.Manolakis,"Introduction To Digital Signal Processing",Macmillan Publishing Company,New York, 1989.

4. "Assembly Programming Using TMS 32010", Texas Instruments, Texas 1989.

5. "VDSP 100 USER'S MANUAL", Vi Microsystems Pvt Ltd.

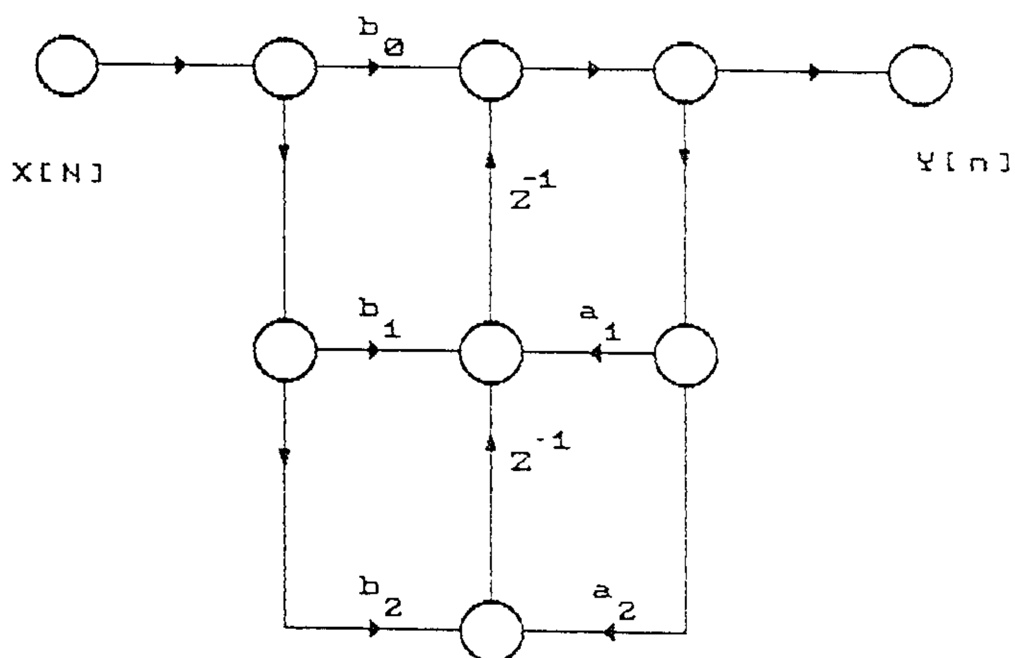
6. "VDSP 100 TECHNICAL REFRENCE ",Vi Microsystems Pvt Ltd.

7. Gottfried, "Programming With C " .Schaum's outline series, Tata Mc Graw Hill,1991.

8. Ray Duncan, Advanced MSDOS Programming,Microsoft Edition,1989.

BASIC STRUCTURE FOR SECOND ORDER

IIR SYSTEMS.



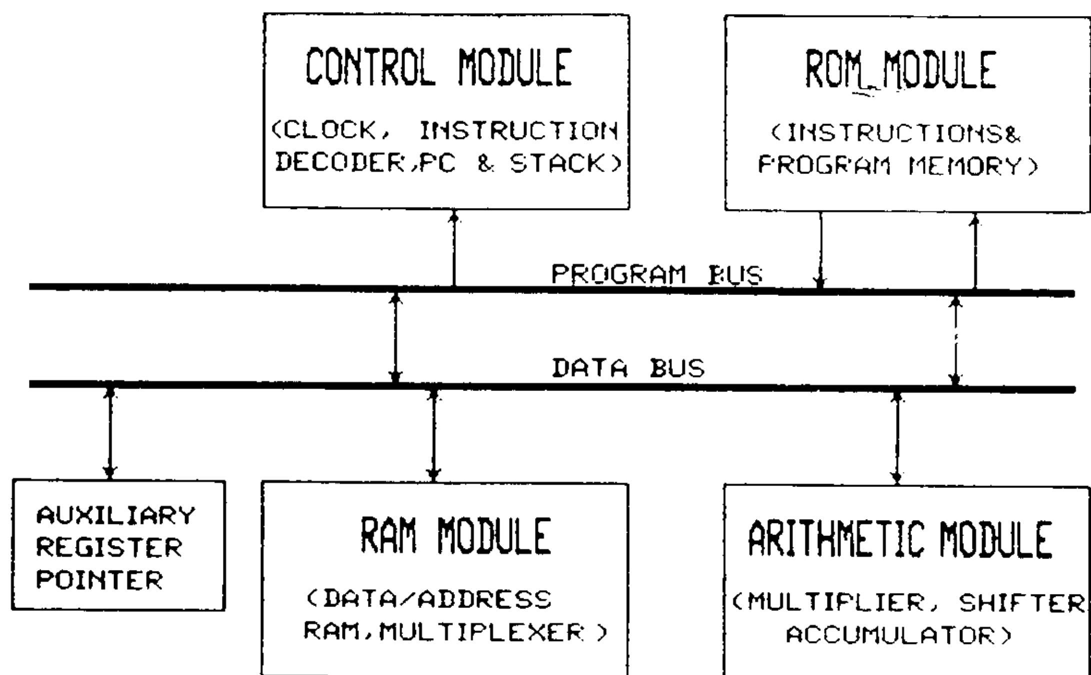
TMS32010

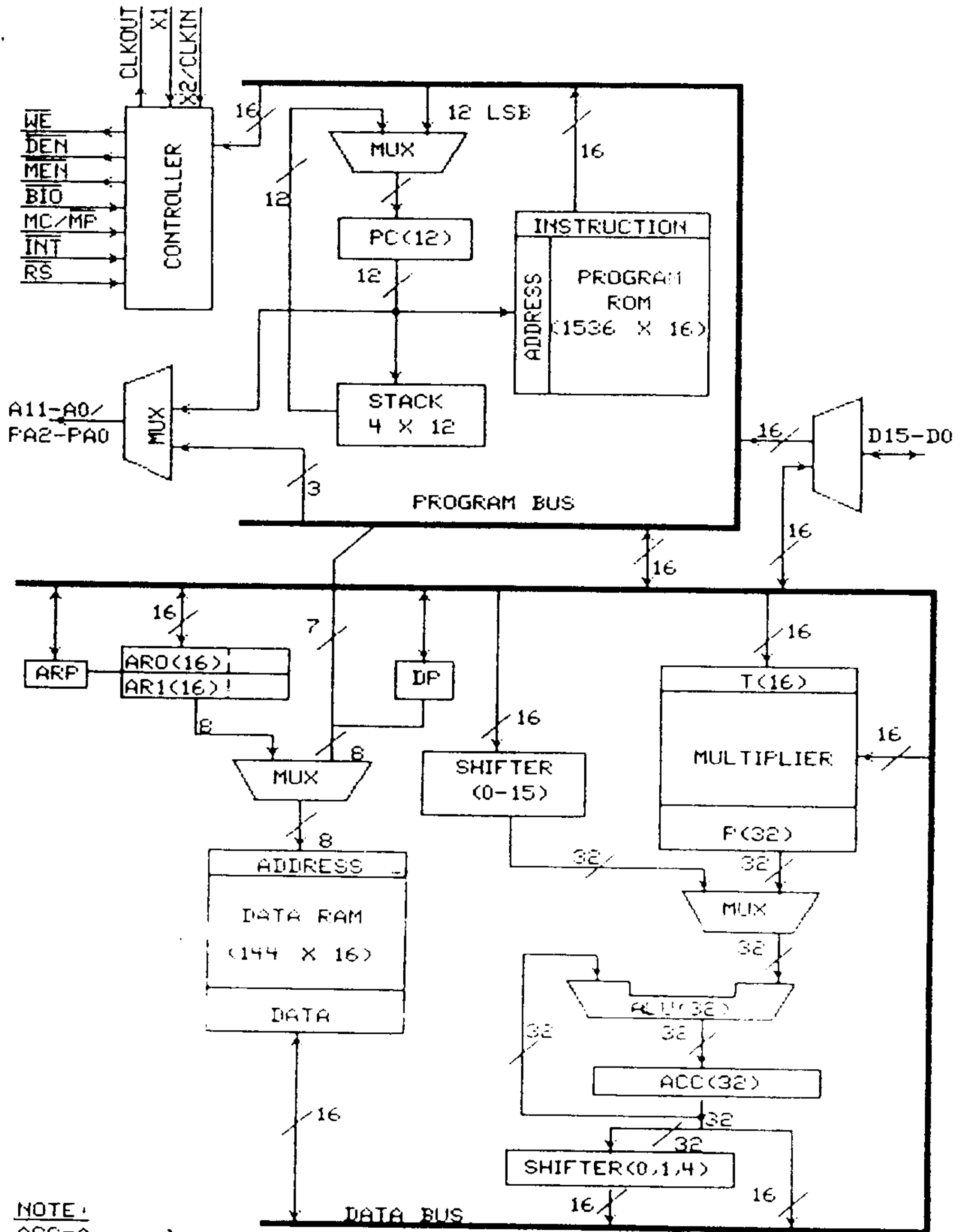
Signal	Pin	Pin	Signal
A1/PA1	1	40	A1/PA1
A1/PA1	2	39	A3
MC/$\overline{MP}$	3	38	A4
$\overline{RS}$	4	37	A5
$\overline{INT}$	5	36	A6
CLKOUT	6	35	A7
X1	7	34	A8
X2/CLK1	8	33	$\overline{MEN}$
$\overline{BIO}$	9	32	$\overline{DEN}$
Vss	10	31	$\overline{WE}$
D8	11	30	Vcc
D9	12	29	A9
D10	13	28	A10
D11	14	27	A11
D12	15	26	D0
D13	16	25	D1
D14	17	24	D2
D15	18	23	D3
D7	19	22	D4
D6	20	21	D5

PIN NOMENCLATURE :-

NAME	I/O	DEFINITION
A11-A0/PA2-PA0	O	External address bus. I/O port address multiplexed over PA2-PA0.
$\overline{BIO}$	I	External polling input for bit test and jump operations.
CLKOUT	O	System clock output, 1/4 crystal/CLKIN frequency.
D15-D0	I/O	16-bit Data bus.
$\overline{DEN}$	O	Data enable indicates the Processor accepting input Data on D15 - D0.
$\overline{INT}$	I	Interrupt.
MC/$\overline{MP}$	I	Memory mode select pin. High selects Microcomputer mode. low selects Micro--processor mode.
$\overline{MEN}$	O	Memory enable indicates that D15-D5 will accept external memory instruction
$\overline{RS}$	I	Reset used to initialize the device.
Vcc and Vss	I	Power and ground pins.
$\overline{WE}$	O	Write enable indicates valid data in.
$\overline{X1}$	I	Crystal input.
$\overline{X2/CLKIN}$	I	Crystal input or external clock input.

SIMPLE BLOCK DIAGRAM





NOTE:
ACC=Accumulator
ARP=Auxiliary Register Pointer
AR0=Auxiliary Register0
AR1=Auxiliary Register1

DP=Data Page Pointer
PC=Program Counter
P=P Register
T=T Register

Figure 2.

MNEMONIC	DESCRIPTION	NO. OF CYCLES	NO. OF WORDS	OPCODE INSTRUCTION REGISTER															
				15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ZAC	Zero accumulator	1	1	0	1	1	1	1	1	1	1	1	0	0	0	1	0	0	1
ZALH	Zero accumulator and load high-order bits	1	1	0	1	1	0	0	1	0	1	I	<—— D ——>						
ZALS	Zero accumulator and load low-order bits with no sign extension	1	1	0	1	1	0	0	1	1	0	I	<—— D ——>						

AUXILIARY REGISTER AND DATA PAGE POINTER INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. OF CYCLES	NO. OF WORDS	OPCODE INSTRUCTION REGISTER															
				15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
LAR	Load auxiliary register	1	1	0	0	1	1	1	0	0	R	I	<—— D ——>						
LARK	Load auxiliary register immediate	1	1	0	1	1	1	0	0	0	R	<—— K ——>							
LARP	Load auxiliary register pointer	1	1	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	K
LDP	Load data memory page pointer	1	1	0	1	1	0	1	1	1	1	I	<—— D ——>						
LDPK	Load data memory page pointer immediate	1	1	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	K
MAR	Modify auxiliary register and pointer	1	1	0	1	1	0	1	0	0	0	I	<—— D ——>						
SAR	Store auxiliary register	1	1	0	0	1	1	0	0	0	R	I	<—— D ——>						

CONTROL INSTRUCTIONS

MNEMONIC DESCRIPTION	NO. OF CYCLES	NO. OF WORDS	OPCODE INSTRUCTION REGISTER															
			15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DINT Disable interrupt	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1
EINT Enable interrupt	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0
LST Load status register	1	1	0	1	1	1	1	0	1	1	1	<——			D			——>
NOP No operation	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
POP Pop stack to accumulator	2	1	0	1	1	1	1	1	1	1	1	0	0	1	1	1	0	1
PUSH Push stack from accumulator	2	1	0	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0
ROVM Reset overflow mode	1	1	0	1	1	1	1	1	1	1	1	0	0	0	1	0	1	0
SOVM Set overflow mode	1	1	0	1	1	1	1	1	1	1	1	0	0	0	1	0	1	1
SST Store status register	1	1	0	1	1	1	1	1	0	0	1	<——			D			——>

BRANCH INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. OF CYCLES	NO. OF WORDS	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
				OPCODE	INSTRUCTION	REGISTER													
B	Branch unconditionally	2	2	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0
				0	0	0	0	<— BRANCH ADDRESS —>											
BANZ	Branch on auxiliary register not zero	2	2	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
				0	0	0	0	<— BRANCH ADDRESS —>											
BGEZ	Branch if accumulator >= 0	2	2	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0
				0	0	0	0	<— BRANCH ADDRESS —>											
BGZ	Branch if accumulator > 0	2	2	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
				0	0	0	0	<— BRANCH ADDRESS —>											
BIOZ	Branch on $\overline{BIO}$ = 0	2	2	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0
				0	0	0	0	<— BRANCH ADDRESS —>											
BLEZ	Branch if accumulator <= 0	2	2	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0
				0	0	0	0	<— BRANCH ADDRESS —>											
BLZ	Branch if accumulator < 0	2	2	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0
				0	0	0	0	<— BRANCH ADDRESS —>											
BNZ	Branch if accumulator = 0	2	2	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
				0	0	0	0	<— BRANCH ADDRESS —>											
BV	Branch on overflow	2	2	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0
				0	0	0	0	<— BRANCH ADDRESS —>											
BZ	Branch if accumulator = 0	2	2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
				0	0	0	0	<— BRANCH ADDRESS —>											
CALA	Call subroutine from accumulator.	2	1	0	1	1	1	1	1	1	1	1	0	0	0	1	1	0	0
CALL	Call subroutine immediately	2	2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
RET	Return from subroutine	2	1	0	1	1	1	1	1	1	1	1	0	0	0	1	1	0	1

T REGISTER, P REGISTER AND MULTIPLY INSTRUCTIONS

MNEMONIC DESCRIPTION	NO. OF CYCLES	NO. OF WORDS	OPCODE INSTRUCTION REGISTER															
			15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APAC Add P register to accumulator	1	1	0	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1
LT Load T register	1	1	0	1	1	0	1	0	1	0	I	<——— D ———>						
LTA LTA combines LT and APAC into one instruction	1	1	0	1	1	0	1	1	0	0	I	<——— D ———>						
LTD LTD combines LT, APAC and DMOV into one instruction	1	1	0	1	1	0	1	0	1	1	I	<——— D ———>						
MPY Multiply with T register; store product in P register	1	1	0	1	1	0	1	1	0	1	I	<——— D ———>						
MPYK Multiply T register with immediate operand; store product in P register	1	1	1	0	0	<——— K ———>												
PAC Load accumulator from P register	1	1	0	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0
SPAC Subtract P register from accumulator	1	1	0	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0

I/O AND DATA MEMORY OPERATION

MNEMONIC	DESCRIPTION	NO. OF CYCLES	NO. OF WORDS	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DMOV	Copy contents of data memory location into next location	1	1	0	1	1	0	1	0	0	1	I	<—			D			—>
IN	Input data from port	2	1	0	1	0	0	0	<–	PA	—>	I	<—			D			—>
OUT	Output data to port	2	1	0	1	0	0	1	<–	PA	—>	I	<—			D			—>
TBLR	Table read from program memory to data RAM	3	1	0	1	1	0	0	1	1	1	I	<—			D			—>
TBLW	Table write from data RAM to program memory	3	1	0	1	1	1	1	1	0	1	I	<—			D			—>

Bits 15–00 are the OPCODE INSTRUCTION REGISTER bits.