Fall 1979

A Microcomputer Implementation of Real Time, Continuously Programmable Digital Filters

William Edward Storma
University of Central Florida, bstorma@bellsouth.net

A MICROCOMPUTER IMPLEMENTATION OF REAL TIME, CONTINUOUSLY PROGRAMMABLE DIGITAL FILTERS

BY

WILLIAM EDWARD STORMA
B.S.E., Florida Technological University, 1978

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in the Graduate Studies Program of the College of Engineering
at the University of Central Florida; Orlando, Florida

Fall Quarter
1979

A MICROCOMPUTER IMPLEMENTATION OF
REAL TIME, CONTINUOUSLY PROGRAMMABLE
DIGITAL FILTERS

BY

WILLIAM E. STORMA

ABSTRACT

When a filter transfer function in s is replaced with the bilinear transform in z, the resulting discrete model represents the original continous model within a second order accuracy of integration. A unique set of recently discovered minimum memory algorithms that perform the bilinear transform on a continuous transfer function are implemented on an INTEL 8080 microprocessor system. Scaling techniques are used to frequency scale all transfer functions to a standardized frequency. All data words are represented in a signed binary double precision format to maintain higher calculation speed and accuracy.

Three test case transfer functions of different order are implemented using the bilinear transform algorithms. First, the algorithms are used to generate the three discrete models. Second, the continuous time models are driven by a step input function, generating a continuous time output. Third, the step function input is discretized and used to drive the bilinear algorithm derived models. Finally, the discrete outputs are compared with the continuous time outputs to validate and evaluate the software techniques used to implement the bilinear algorithms, which imply that the techniques provide a basis for new hardware designs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

Analog circuits and filters designed to process analog signals often are limited in accuracy due to:

a. thermal drift
b. component tolerances
c. offset and bias conditions of operational amplifiers
d. signal noise introduced by the circuit itself

The only means to build highly accurate analog circuits is through careful design and the use of high quality components. This often results in designing expensive circuits and allowing bench time to minimize circuit sensitivities due to circuit parameters.

The age of digital electronics has brought about many new methods to handle the processing of analog signals. The ability to design signal processing circuits that can handle the signals digitally overcomes many of the handicaps of the analog circuits. Digital Signal Processing (D.S.P.) is a newer, more accurate and less expensive means to analyze and process signals. The digital circuits have no thermal drift, no offset or bias problems, do not require high quality circuit components, and do not introduce noise into the circuits. Thus, many signal processing systems have become digital in nature, using analog-to-digital (A/D) and digital-to-analog (D/A) converters to interface between the analog and digital systems.

The design of digital filters, a special case of D.S.P., has become a fairly common practice with standardized design procedures.

The use of these standard design procedures involves implementing a filter transfer function in the form of a difference equation. The result of this design is a digital circuit that is 'hard wired', i.e. the characteristics of the circuit are not readily alterable. This feature is unfortunate if the exact characteristics of the filter are unknown and several designs must be tried before a circuit is chosen.

An alternative to the above problem is the design of a computer software package that allows a real time implementation of a filter transfer function 'in circuit'. Also, giving the software package the ability to alter the filter transfer function while the digital filter is processing signals allows a 'continuous programming' feature. The result is a real time continuously programmable digital filter. By using an interface capability, the software can be implemented on a microprocessor system and run 'in circuit'. This allows the microprocessor to actually synthesize any filter function and modify the transfer function characteristics while the filter is 'in circuit'.

The basis of this thesis is the implementation of a software package as described above. The software package is designed around a new set of algorithms that perform a bilinear transform using a minimum memory approach. An INTEL 8080/8085 based microprocessor is used to process these bilinear algorithms. The program starts with a transfer function in differential equation (or s domain) form. Then, using a bilinear transform approach, the differential equation is transformed into a difference equation. The program then executes the difference equation in a real time mode, allowing real time output.

The program has memory allocated to operate on transfer functions up to fifth order, using a double-precision (16 bit) data word. The output from the program is a transient response in time, with the input presently being a step function (though easily modified for any signal input). A transient response (or time response) is preferred over a frequency response in this case since a step function inputted in a transfer function forces all filter characteristics to be displayed in the output. The combined features of a digital filter that is continuously programmable, operates in real time, and can be used 'in circuit' make this digital filter system highly useful in the design of digital signal processing systems.

II. BACKGROUND

Filtering is a technique whereby the frequency spectrum of a signal is specified, such that certain frequencies are passed through the filter and other frequencies are rejected by the filter. Filters are initially designed in the frequency domain (or complex s plane), where the frequency characteristics can be used to obtain a differential equation. This characteristic filter equation is usually refered to as a transfer function (denoted by H(s)) and is a ratio between the output (Y(s)) and the input (X(s)). The equation is written as:

$$\frac{Y(s)}{X(s)} = H(s) \qquad 2.1)$$

and is described in the block diagram form as:

$$X(s) \longrightarrow \boxed{H(s)} \longrightarrow Y(s)$$

where

$$Y(s) = H(s)\ X(s)$$

Once an H(s) is specified, the equation can be transformed into the time domain, using an inverse Laplace transform:

$$\mathcal{L}^{-1}[\ H(s)\ ] = h(t) \qquad 2.2)$$

The resulting h(t) is an equation of the analog filter characteristics in a continuous time domain. Analog filter design, unlike digital filter design, can be run on an analog computer, which operates in a continuous time mode. However, with the advent of high speed digital computers, a trend has developed to use digital equipment to implement algorithms. The digital computer requires that the algorithms be modified to work in other than a continuous time domain. This is because a digital computer does not run in a continuous time mode, like the analog computer, but in a discrete time mode. This discrete time mode is due to the fact that a digital computer works in cycle times, and calculations require a certain number of machine cycles to implement. The result from a digital computer is a string of outputs at discrete intervals of time.

It is therefore necessary to transform an H(s) into a discrete time mode equation. The necessary discrete time mode equation is the difference equation, which is implemented in the z domain. The equation is written as:

$$\frac{Y(z)}{X(z)} = H(z) \qquad 2.3)$$

where X(z) are discrete time inputs and Y(z) are discrete time outputs. The transformation from the z domain to a discrete time mode, nT, is called the inverse z transform, denoted by:

$$\mathcal{Z}^{-1}[\ H(z)\ ] = h(nT) \qquad 2.4)$$

where T is the time sample interval and n is the n^{th} sample period.

Ordinarilly, H(s) models are not transformed directly to H(z) models. As an example of a textbook approach, the H(s) must first be transformed into an h(t), then the continuous time, t, must be changed to a sample interval time, nT, and finally the h(nT) must be transformed to an H(z).

Mathematically:

$$h(t) = \mathcal{L}^{-1}[\ H(s)\ ] \tag{2.5}$$

$$h(nT) = h(t)\Big|_{t\ =\ nT} \tag{2.6}$$

$$H(z) = \mathcal{Z}[\ h(nT)\ ] \tag{2.7}$$

This and other similar approaches are cumbersome and slow processes for a digital computer to perform. What would be more desireable would be an algorithm that could calculate an H(z) based on an H(s). This would avoid having to transform into and out of the time domain. This calculation for an s to z conversion would be an approximation of H(z), based on H(s) and sampling rates.

Although there are computer programs for transforming from the s to the z domain, these programs require some amount of memory for all temporary results. Some digital systems posess only a small memory and therefore cannot use the s to z transformation processes. What would be ideal for these digital systems with small memory space would be an accurate algorithm that could approximate an H(z), based on an H(s) and the sampling rate, and perform this algorithm 'in place', i.e. using only the memory required for coefficient storage for the algorithm process.

The specific algorithm to be discussed is based on the bilinear transform:

$$s = \frac{2}{T}\left(\frac{z-1}{z+1}\right) \tag{2.8}$$

which is the average of the first order forward difference equation and the first order backward difference equation. This bilinear transform is the standard algorithm used in digital filter design.

The in-place algorithms for equation 2.8 were discoverd in 1978 [3] and were published and later modified to handle any general bilinear transformation [2]. The general form of the algorithms are reprinted here for convenience:

the bilinear transform: $s = \frac{az+b}{cz+d} = \frac{\alpha}{z+\beta} + \gamma \; ; \quad c \neq 0$ 2.9)

where $\gamma = \frac{a}{c} \qquad \beta = \frac{d}{c} \qquad \alpha = \frac{b}{c} - \beta\gamma$

and the $\frac{2}{T}$ factor is incorporated into the a,b,c,d variables. Now, given a polynomial in z:

$$D(z) = \sum_{i=0}^{N} d_i \, z^i \qquad 2.10)$$

and the bilinear transform (equation 2.9), the polynomial D(s) is found by:

$$D(s) = \sum_{i=0}^{N} d_i \left(\frac{az+b}{cz+d}\right)^i = \frac{P(z)}{(cz+d)^N} \qquad 2.11)$$

or

$$P(z) = \sum_{i=0}^{N} p_i \, z^i = (cz+d)^N \, D(s) \qquad 2.12)$$

The problem in getting an 'in place' algorithm requires computing the p_i's, the coefficient set of P(z), from the d_i's, the coefficient set of D(s).

The four step algorithm process for this bilinear transformation is as follows:

substituting 2.9 into 2.12:

$$P(z) = c^N \, (z+\beta)^N \, D\left(\frac{\alpha}{z+\beta} + \gamma\right) \qquad 2.13)$$

Equation 2.14 can be broken down into elementary transforms, which are:

$$E(z) = D(z+\gamma) \qquad 2.14)$$

$$F(z) = c^N E(2z) \tag{2.15}$$

$$G(z) = z^N F(1/z) \tag{2.16}$$

$$H(z) = G(z+\beta) \tag{2.17}$$

Each elementary transform consists of a shift in the z domain of the form:

$$z = 2z \tag{2.18a}$$

$$z = z+\beta \tag{2.18b}$$

$$z = 1/z \tag{2.18c}$$

$$z = z+\gamma \tag{2.18d}$$

and each of these operations can be applied to polynomials by an 'in place' operation. This means that any bilinear transform can be applied to polynomials by performing a sequence of 'in place' operations, such as the general equations of 2.18.

To prove that $H(z) = P(z)$, substitute 2.16 into 2.17, 2.15 into 2.16 and 2.14 into 2.15.

$$\begin{aligned} H(z) &= G(z+\beta) \\ &= (z+\beta)^N F\left(\frac{1}{z+\beta}\right) \\ &= (z+\beta)^N c^N E\left(\frac{\alpha}{z+\beta}\right) \\ &= (z+\beta)^N c^N D\left(\frac{\alpha}{z+\beta} + \gamma\right) \\ &= P(z) \end{aligned} \tag{2.19}$$

The strategy is to compute first the coefficients of $E(z)$ from the coefficients of $D(s)$, then the f_i's from the e_i's, then the g_i's from the f_i's and finally the h_i's from the g_i's.

From these elementary transforms, a set of computational equations can be obtained [2]. The final form of these equations are:

$$e_j = d_j + \sum_{i=j+1}^{N} \binom{i}{j} \gamma^{i-j} d_i \qquad 2.20)$$

$$f_i = c^N \alpha^i e_i \qquad 2.21)$$

$$g_i = f_{N-i} \qquad 2.22)$$

$$h_j = g_j + \sum_{i=j+1}^{N} \binom{i}{j} \beta^{i-j} g_i \qquad 2.23)$$

where

$$\binom{i}{j} = \frac{i!}{j!(i-j)!}$$

An analysis of these equations will prove that all these operations can be performed 'in place'. For the general case of a transfer function in H(s):

$$H(s) = \frac{\sum_{i=0}^{M} a_i s^i}{\sum_{i=0}^{N} b_i s^i} = \frac{A(s)}{B(s)} \qquad 2.24)$$

the four step bilinear algorithm would be applied to both the numerator and the denominator seperately, with the highest coefficient order (either M or N) being the order of both the numerator and denominator in H(z). The H(z) would then be written as (assuming M^{th} order):

$$H(z) = \frac{\sum_{i=0}^{M} c_i z^i}{\sum_{i=0}^{M} d_i z^i} = \frac{C(z)}{D(z)} \qquad 2.25)$$

The resulting coefficients of H(z), i.e. the c_i's and d_i's, now occupy the memory locations originally designated for the

a_i's and b_i's, respectively. After obtaining the H(z), an inverse z transform can be applied to transform the equation to the time domain. For the general case:

$$H(z) = \frac{c_m z^m + c_{m-1} z^{m-1} + \cdots + c_0 z^0}{d_m z^m + d_{m-1} z^{m-1} + \cdots + d_0 z^0} \qquad 2.26)$$

which can be rearranged as follows:

$$X(z)[\ c_m z^m + c_{m-1} z^{m-1} + \cdots + c_0 z^0\ ] =$$
$$Y(z)[\ d_m z^m + d_{m-1} z^{m-1} + \cdots + d_0 z^0\ ] \qquad 2.27)$$

Applying the inverse z transform, the equation becomes:

$$c_m x(nT+mT) + c_{m-1} x(nT+(m-1)T) + \cdots + c_0 x(nT) =$$
$$d_m y(nT+mT) + d_{m-1} y(nT+(m-1)T) + \cdots + d_0 y(nT) \qquad 2.28)$$

The inputs (x(nT+iT)) and the outputs (y(nT+iT)) both depend on values at time t=nT and all future time values (t=nT+T, nT+2T, ⋯). The equation can be converted so that the inputs and outputs depend only on present (t=nT) and past values of time (t=nT-T, nT-2T, ⋯). This can be accomplished by allowing

$$n = n-i \qquad 2.29)$$

where n is the n^{th} coefficient. This amounts to a shift in time. The difference equation now becomes:

$$c_m x(nT) + c_{m-1} x(nT-T) + \cdots + c_0 x(nT-mT) =$$
$$d_m y(nT) + d_{m-1} y(nT-T) + \cdots + d_0 y(nT-mT) \qquad 2.30)$$

The output at present time, y(nT), can be expressed as a function of the present input and all past inputs and outputs of the equation, as follows:

$$d_m y(nT) = c_m x(nT) + c_{m-1} x(nT-T) + \cdots + c_0 x(nT-mT) -$$

$$d_{m-1} y(nT-T) - \cdots - d_0 y(nT-mT) \qquad 2.31)$$

which can be rewritten as:

$$y(nT) = \frac{\sum_{i=0}^{M} c_{M-i}\ x(nT-iT) - \sum_{i=1}^{M} d_{M-i}\ y(nT-iT)}{d_M} \qquad 2.32)$$

The equations necessary to perform a bilinear transformation on an H(s) have been developed. Also, the necessary equations have been developed that will output a string of values based on a string of input values. What has been derived is a set of equations that allows a programmable implementation of a digital filter on a digital computer. By a proper adjustment of the output rate of the string of values from equation 2.32, the input-output operation could be performed in a 'real time' mode. By updating the original H(s) equation and allowing the bilinear transform to compute a new H(z), the digital filter could become ' continuously programmable' and run in 'real time'.

The implementation of the above bilinear transform algorithm and a corresponding input-output routine are discussed in the following sections. The implementation is a direct result of the equations developed in this section.

III. DATA FORMAT CONSIDERATIONS

Implementation of the bilinear transform algorithm on an 8 bit microcomputer poses some questions as to how the software is to handle the program data. The areas of concern in dealing with the data handling problems are:

a. should the program use fixed point binary or floating point binary?
b. should the program use single or double precision?
c. what is the highest order transfer function that can be implemented, with respect to points a and b.

These are the software data handling problems that must be answered before the actual software programs can be written.

The first data handling question concerns the method of representing the data during algebraic manipulations. The use of floating point notation allows data to be described over a wide range of values. Floating point notation has a unique data structure and cannot be represented with a normal 8 or 16 bit data word. Due to the long data word required for floating point notation, execution times for floating point routines are excessively long when compared to analagous routines that are performed in a fixed point notation. Since a requirement in executing these transform algorithms is a rapid execution speed, the use of any floating point notation would cause a considerable increase in the total execution time of a program, which is a feature that cannot be tolerated in executing these routines. Another disadvantage of using a floating point notation is that the number of bits allocated for the data (mantissa) are not the full 16 bits that are used in the double precision fixed point notation. This means that the floating point notation will not carry a full 16 bit accuracy in data and therefore is less accurate than the fixed point notation in describing data. This factor reinforces the undesireable aspects of using floating point notation.

This leaves the fixed point representation of data to be considered. Using a signed binary notation, data can be ranged over ± 127 for single bit precision and ranged over ± 32767 for double precision. If the sign bit is stored somewhere else than with the data, the double precision data could be ranged over ± 65535. In all cases, all integer values can be accounted for in the fixed point representation. There still exists a problem in describing data that exists in a fractional form or has some part of the data in fractional form (i.e. 123.78, where the .78 is the fractional part). To use data in fractional form, all the data can be scaled to a pure fractional form (i.e. all data ranged between -1 and +1, excluding endpoints). This can be accomplished by dividing all the data by a value, R, which is greater in magnitude than any of the data, to convert all the data to a fractional form.

The result of scaling all the data to be less than the magnitude of one provides a method of describing all data combinations with a high degree of accuracy. For a single precision notation, numbers as small as 2^{-8} (3.90625×10^{-3}) can be described and for double precision notation, numbers as small as 2^{-16} (1.525×10^{-5}) can be described. In both of the above fractional cases, it is assumed that the sign bit is carried elsewhere and is not part of the 8 or 16 bit data word. Therefore, by properly scaling all of the data to a fractional form, the accuracy of the data can be maintained.

From all the information known about fixed point binary and floating point binary data, and the knowledge that the bilinear transform algorithm requires rapid machine algebraic computations and accurate data handling, one can postulate that the fixed point binary data technique is best. To maintain the high accuracy of the data during the algebraic computations, a 16 bit double precision fractional format is necessary. To maximize the data accuracy, the sign bit of the double precision data word is stored elsewhere than with the data word itself.

Having answered the data handling questions to the first and second areas, there remains the question as to what is the highest order transfer function that can be implemented. With the knowledge that double precision fixed point notation is used, it is necessary to determine what is the smallest data word that can be accurately described. Part of this question can be quickly determined by examining the bilinear transform. An examination of equation 2.21, which is:

$$f_i = c^N \alpha^i e_i \tag{3.1}$$

depicts that the α is raised to a power, i, which is directly related to the order of the e coefficient. For the bilinear transform of

$$s = \frac{z-1}{z+1} \qquad 3.2)$$

with the $\frac{2}{T}$ factor set equal to one, the value of α becomes

$$\alpha = \frac{b}{c} - \beta\gamma = -1 - 1 = -2 \qquad 3.3)$$

and

$$c = 1$$

With this information, equation 3.3 becomes

$$f_i = (-2)^i\ e_i \qquad 3.4)$$

For an N^{th} order system, the e_N coefficient would be multiplied by a $(-2)^N$ value. To insure that the f_N coefficient be less than the magnitude of one, the e_N can be divided by a 2^{N+1}.

There still exists the problem of a data overflow in equations 2.20 and 2.22, due to the summations. Since the summed value is determined by all the higher order factors and these higher order factors can range in value between $\pm$ 1, there is no absolute factor to divide all the data by to insure against an overflow. Therefore, it was necessary to determine a scaling factor based on sample problems. By inspection of these sample problems and extrapolation of the scaling factors determined for these sample problems, an overall data scaling factor of 2^{2N-1} has been determined for all realizeable filter functions. From the data scaling factor and the need to maintain some degree of accuracy in the data, an initial limit on transfer functions has been determined to be fifth order. Using the double precision fixed point notation, the data would be maximally scaled by 2^9 (512), which leaves, at most, seven bits of data that can be retained after the scaling process.

Based on the information presented and the knowledge of the bilinear transform algorithm, filter transfer functions should be no greater than fifth order. This allows sufficient data accuracy for the double precision fixed point binary data format, which is to be used in the algebraic computations. The basic questions as to what data handling techniques the software should use have been answered. The next step is to scale the differential equation for use by the bilinear transform.

IV. SCALING THE FILTER FUNCTION

Any given filter transfer function in differential equation form will contain coefficients for each power of s. For any general case, the coefficients will be any real number. These coefficients must be converted to a double precision fixed point fractional binary number before being implemented. Therefore, the transfer function coefficients must all be scaled prior to implementing the bilinear transform algorithm. A generalized scaling technique must be obtained to handle any general transfer function.

Based on a bilinear transform of equation 3.2, a scaling factor of 2^{2N-1} was determined necessary to prevent data overflow during the bilinear transform algorithm. This scaling factor was determined with the $\frac{2}{T}$ factor set equal to one. In general, the $\frac{2}{T}$ factor is not equal to one and must be accounted for. If the $\frac{2}{T}$ factor were to be included in the a,b,c,d of equation 2.9, then equation 3.2 would really be expressed as:

$$s = \frac{2z - 2}{Tz + T} \qquad 4.1)$$

and the α, β, γ factors would all be influenced by T. Due to this influence by T, the α, β, γ factors would have to be changed every time a different T is chosen. Since the α, β, γ factors must be included in the bilinear transform, the software must be alterable to handle the changes in α, β, γ.

The variations in α would complicate the implementation of equation 3.4, since raising a number α to a power is not easily done on a microprocessor. However, raising 2 to a power can be quickly accomplished on binary data by a sequence of shift operations. Therefore, it would be convenient to keep the $(-2)^i$ factor in equation 3.4 . It is therefore necessary to scale the transfer function to redefine the $\frac{2}{T}$ factor to be equal to one.

The T factor must first be related to the filter frequency. Consider a filter with a natural frequency of ω. The period of this filter is then τ . The T factor is then some fractional part of τ, such that an integral multiple of T will equal τ. This integral multiple can be defined as x and is called a sample interval. Now, to obtain $\frac{2}{T} = 1$, a frequency scaling technique must be incorporated. Given a sample interval, x, which determines the number of data outputs (from the difference equation) per period, the original transfer function (at ω) yields:

$$\omega = 2\pi f \qquad \tau = \frac{2\pi}{\omega} = \frac{T}{x} \tag{4.2}$$

Therefore:

$$\frac{2}{T} = \frac{2}{x\tau} = \frac{2\omega}{2\pi x} \tag{4.3}$$

Now, consider scaling the frequency to some ω', such that $\frac{2}{T} = 1$. Under these conditions:

$$\frac{2}{T} = \frac{2}{x\tau'} = \frac{2\omega'}{2\pi x} \tag{4.4}$$

or

$$x = \frac{2\omega'}{2\pi} \tag{4.5}$$

To frequency scale from ω to ω', substitute equation 4.5 into equation 4.1, as shown:

$$\frac{2\omega}{2\pi x} = \frac{2\omega'}{2\pi \frac{2\omega'}{2\pi}} = \frac{\omega}{\omega'} \qquad 4.6)$$

which can be rewritten as:

$$\omega' \frac{\omega}{\pi x} = \omega \qquad 4.7)$$

Equation 4.7 is the factor necessary to frequency scale from ω to ω'. By using this scaling format, the $\frac{2}{T}$ factor will always be set equal to one. For a general polynomial in s, the coefficients are scaled using the formula:

$$p_i' = \left(\frac{\pi x}{\omega}\right)^{N-i} p_i \qquad 4.8)$$

For a normalized polynomial, with $\omega > \pi x$, the coefficients of P(s) are scaled down to a fractional value, with the exception of the N^{th} coefficient, which is one. Once all the polynomial coefficients are in a frequency scaled form, the additional scaling factor of 2^{2N-1} can be performed. The generalized scaling algorithm now becomes:

$$p_i' = \frac{\left(\frac{\pi x}{\omega}\right)^{N-i}}{2^{2N-1}} p_i \qquad 4.9)$$

This scaling algorithm insures that all the coefficients are properly scaled to a fractional value and will not overflow during the bilinear transform algorithm process.

V. SOFTWARE IMPLEMENTATION

Knowing the necessary equations to perform the bilinear transform (equations 2.20 - 2.23) and that the data is to be represented in a double precision fixed point signed binary format, the actual software programming can be implemented. Knowledge of the bilinear transform equations only describes the algorithm, but does not specify how the equations are to be implemented in a software program. These implementation procedures are based on the programmers' interpretation of the equations and his experience of using a particular programming language.

Based upon the transfer function limit of fifth order and the full 16 bit data word, certain initial configurations for memory storage locations are possible. The data is stored as two 8 bit words with a third 8 bit word storing the sign bit, described as follows:

M	M.S.B.
M + 1	L.S.B.
M + 2	Sign byte

with M.S.B. denoting most significant byte and L.S.B. denoting least significant byte. Only one bit of the sign byte is used, with the other bits set to zero. For positive numbers, bit 7 is set to zero and for negative numbers bit 7 is set to one. Since three memory locations are necessary to fully describe a data word and a fifth order polynomial can have six coefficients (0 - 5), there must be eighteen memory storage locations to store all the coefficients of a fifth order polynomial. A filter transfer function could possibly exist as a fifth order numerator over a fifth order denominator, therefore a total of thirty two memory locations are needed to store the coefficients of a transfer function in memory.

Knowing that the bilinear transform is to be performed on data 'in place', then once the transform algorithms are executed, the coefficients stored in the memory locations for the transfer function now store the coefficients for the difference equation. The inverse z transform then allows the coefficients of the difference equation to become the coefficients of the discrete time equation. Since every coefficient of a discrete time equation must have a discrete time factor associated with it (i.e. $p(nT-iT)$), there must be six discrete time factors each for the numerator and denominator discrete time equations. The discrete time factors are also described using the double precision fixed point signed binary format that is used on the transfer function coefficients. This requires another thirty two memory locations to store these discrete time factors. On the basis of this requirement for memory, an allocation for memory space was chosen, as shown in figure 1.

The next step involves implementing the bilinear transform equations (equations 2.20 - 2.23). One of the first questions is concerned with implementing the binomial factor

$$\binom{i}{j} = \frac{i!}{j!(i-j)!} \qquad 5.1)$$

Equation 5.1 can either be calculated each time equation 2.20 or

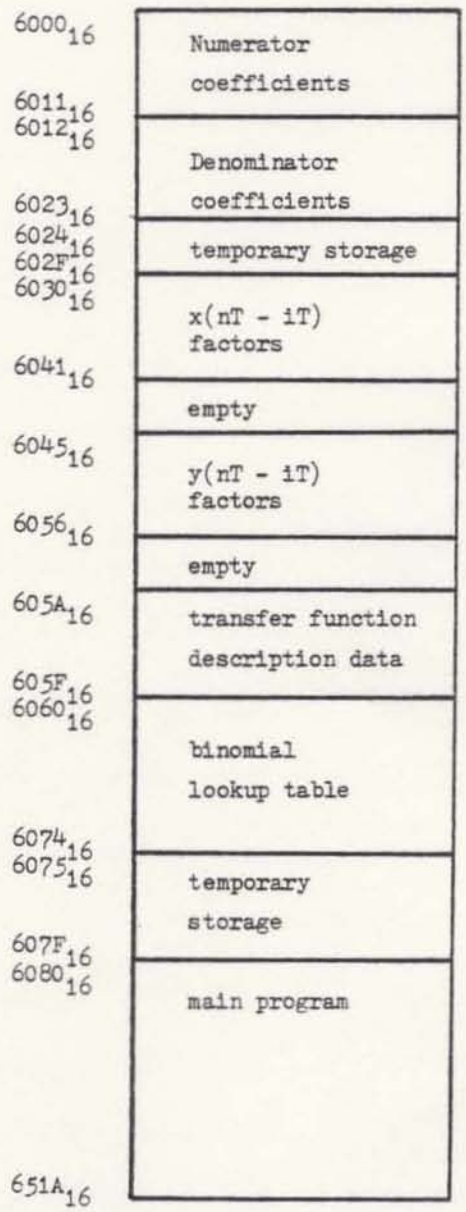

Figure 1. Memory map of data storage

2.23 is performed, or a lookup table, based on i and j, could be performed. Knowing that rapid computations are desired and that a factorial computation requires repeated multiplication, which requires an extensive amount of computer computation time, a lookup table would be easier to implement and faster to execute. To implement the lookup table, a means to uniquely describe every i and j combination must be determined. Examination of equations 2.20 and 2.23 show that i is less than j for all cases of i. These restrictions state that some combinations of i and j do not occur in these equations and can be disregarded. A means to determine a number that is unique for all the possible combinations of i and j is to multiply i and j such that

$$K = i \times j \qquad 5.2)$$

This K value can then be used to locate the position in memory of the proper binomial value. The binomial number can then be retrieved and used in the proper transform equation. The binomial lookup table, based on equations 5.1 and 5.2, is shown in figure 2. The value of K is added to memory location 6060_{16} to 'point at' the binomial value to be retrieved from the table.

To implement the bilinear transform equations (2.20 - 2.23), a structured programming method is a desireable choice, both to aid in understanding the flow of the program and to break the transform process into 'blocks' that perform a specific equation on a specific section of data. Equations 2.20 - 2.23 must be performed on both the numerator and denominator coefficients. Therefore, a software subprogram must be written for each transform equation twice, once

6060_{16}	1
	1
	2
	3
	4
	5
	3
	3
	6
	6
	10
	10
	4
	4
	4
	10
	10
	10
	10
	10
6074_{16}	5

Figure 2. Binomial lookup table

for the numerator coefficients and once for the denominator coefficients.

The first equation to be implemented is equation 2.20, which is:

$$e_j = d_j + \sum_{i=j+1}^{N} \binom{i}{j} \gamma^{i-j} d_i \tag{5.3}$$

Using the bilinear transform of

$$s = \frac{z-1}{z+1} \tag{5.4}$$

with

$$\frac{2}{T} = 1 \tag{5.5}$$

the factors α, β, γ become

$$\alpha = -2 \tag{5.6a}$$
$$\beta = +1 \tag{5.6b}$$
$$\gamma = +1 \tag{5.6c}$$

Equation 5.3 reduces to

$$e_j = d_j + \sum_{i=j+1}^{N} \binom{i}{j} d_i \tag{5.7}$$

A flow chart depicting the implementation of equation 5.7 on the numerator and denominator coefficients is displayed in figures 3 and 4, respectively. In both subprograms (XFORM and XFORM2), the program starts at $j=0$, evaluates the binomial factor and sums the partial products onto e_j . Once $i=N$, j is incremented and the process repeats itself until $j=N$. The value of N is stored in the memory as NUM for the numerator and DEN for the denominator. These values must be placed in memory before the transformation process begins. Once $j=N$, equation 5.7 will have been implemented on all the coefficients and the program moves on to the next

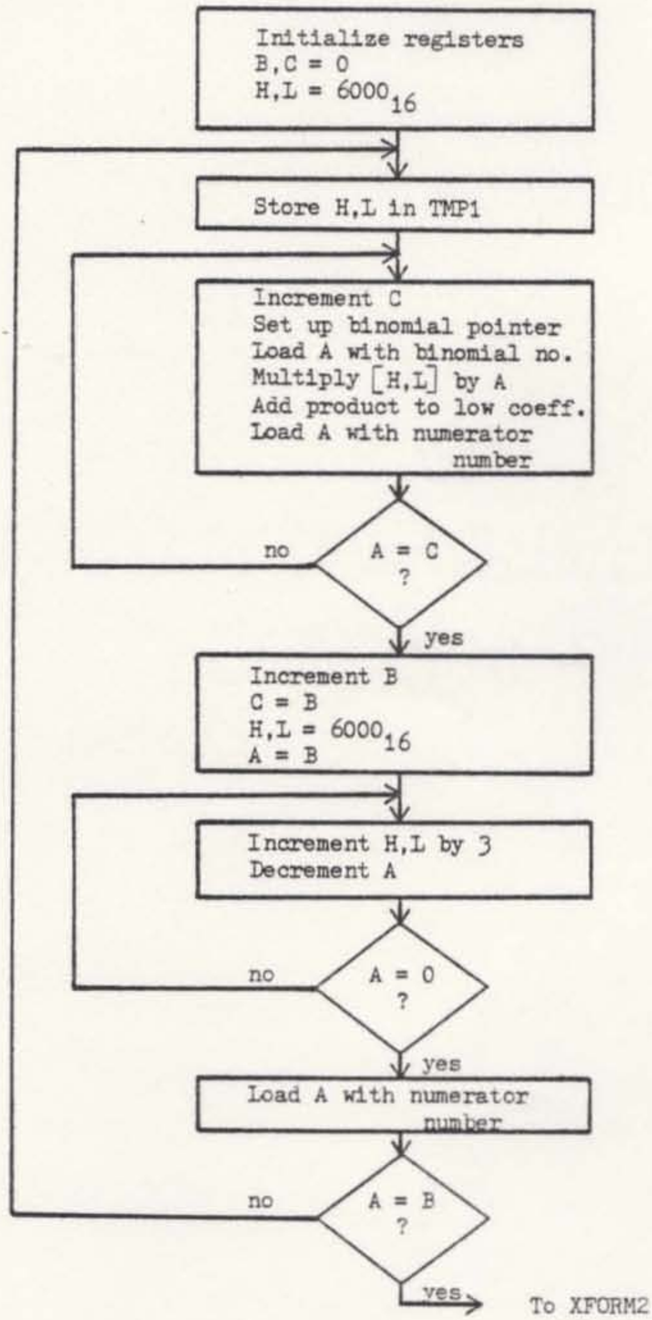

Figure 3. XFORM flow chart

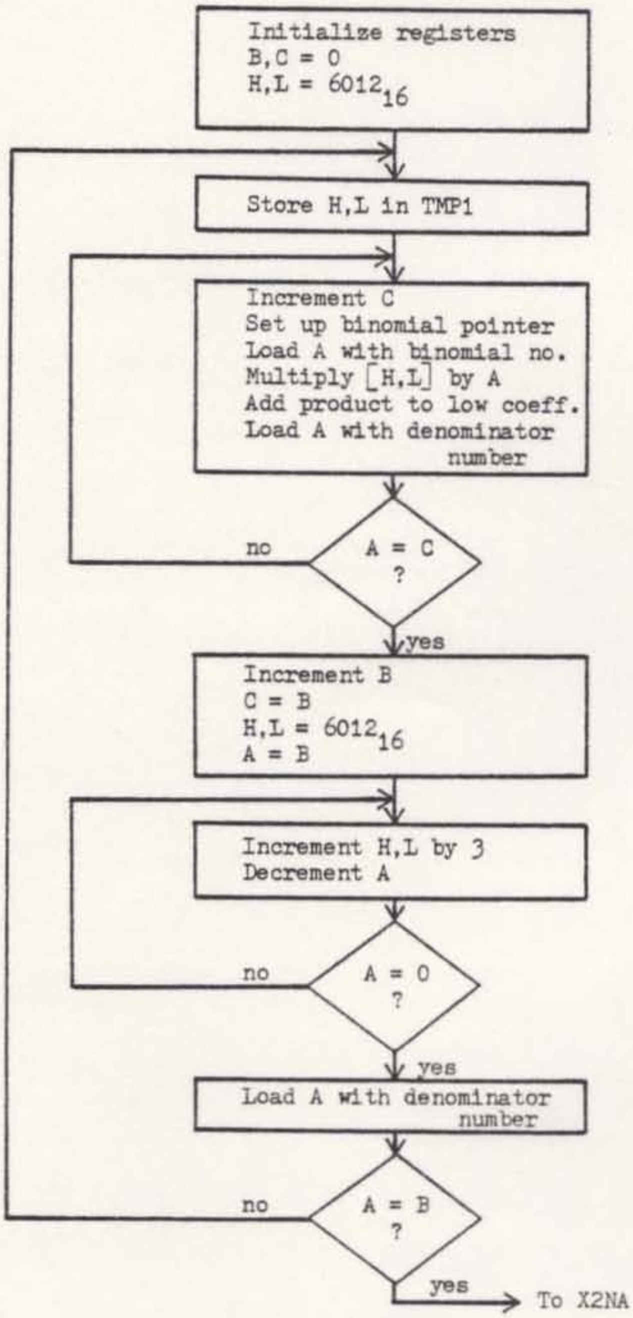

Figure 4. XFORM2 flow chart

subprogram.

The next equation to be implemented is equation 2.21, which is:

$$f_i = c^N \alpha^i e_i \qquad 5.8)$$

which reduces to:

$$f_i = (-2)^i \; e_i \qquad 5.9)$$

Equation 5.9 can be very easily implemented on a microcomputer. Any multiplication by two can be performed by a series of shift operations. A flow chart implementing equation 5.9 on the numerator and denominator coefficients is shown in figures 5 and 6, respectively. Again, the subprogram (X2NA or X2NB) starts with i=0, performs equation 5.9 and then increments i, repeating equation 5.9 until j=N, when the process is finished. The program then proceeds to the next subprogram.

The third equation to be implemented is equation 2.22, which is:

$$g_i = f_{N-i} \qquad 5.10)$$

This equation redefines the order of the coefficients. By keeping track of where all the coefficients are for both the numerator and denominator, the reassignment of the coefficients can be handled with software programming. This means that equation 5.10 does not have to be actually performed. This allows a saving of computation time since equation 5.10 is not actually implemented and this helps to reduce the total execution time of the program.

The last equation to be implemented is equation 2.23, which is:

$$h_j = g_j + \sum_{i=j+1}^{N} \binom{i}{j} \beta^{i-j} \; g_i \qquad 5.11)$$

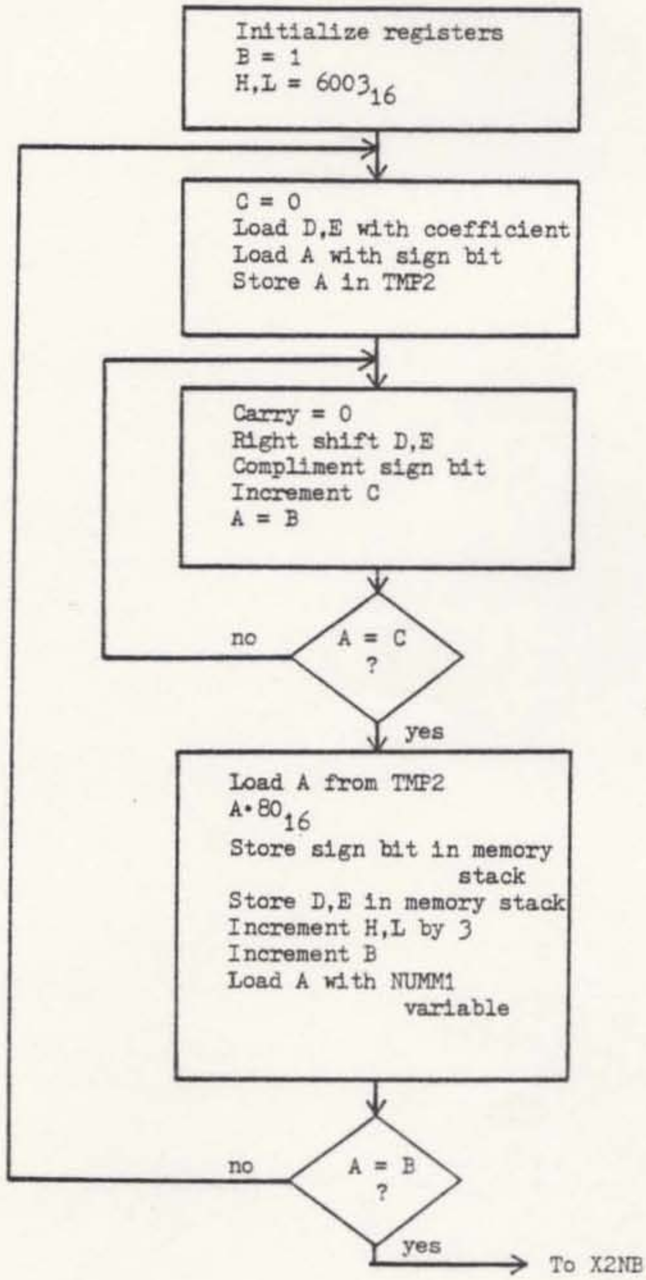

Figure 5. X2NA flow chart

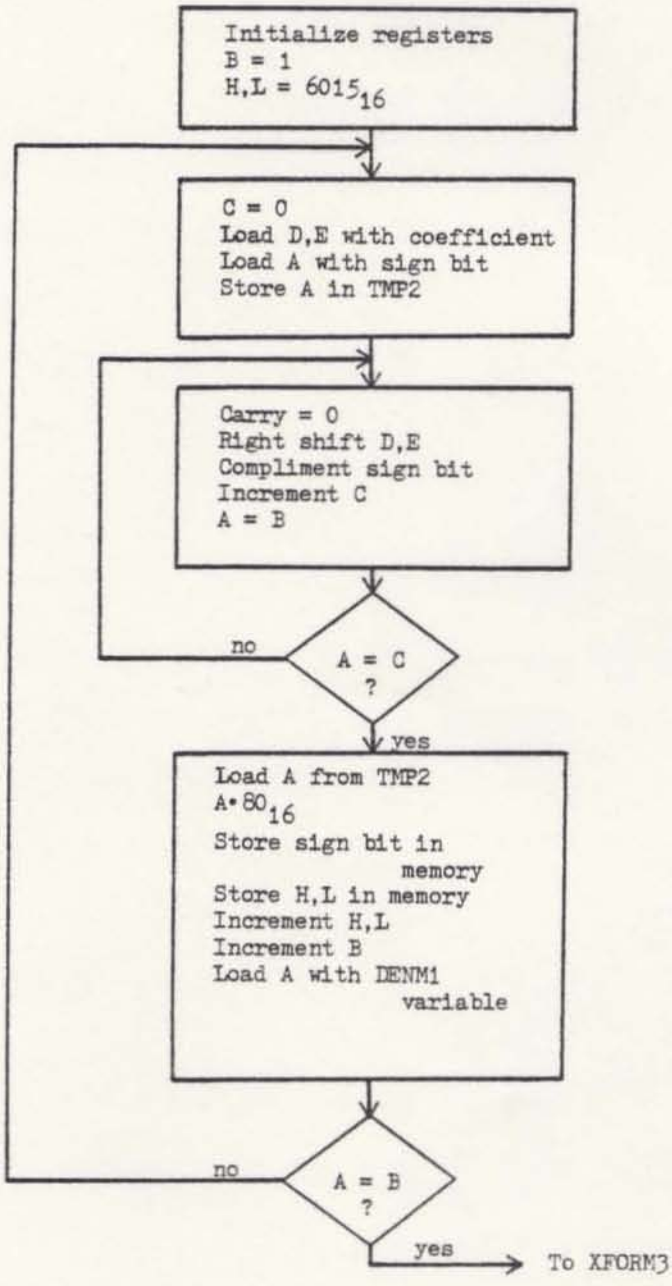

Figure 6. X2NB flow chart

This equation can be reduced to:

$$h_j = g_j + \sum_{i=j+1}^{N} \binom{i}{j} g_i \qquad 5.12)$$

since $\beta=1$. Equation 5.12 is identical to equation 5.7 in form, so the actual programming should be similar. However, equation 5.12 must be executed on coefficients that have been reversed in order. This difference must be accounted for in the subprogram (XFORM3 and XFORM4). Figure 7 and 8 depict the flow charts of the subprograms that operate on the numerator and denominator coefficients, respectively.

Thoughout the subprograms that implement the bilinear transform, certain variables are used to allow the program to know the order of the transfer functions and properly implement the subprograms. These variables are dependent on the order of the transfer function and are obtained by using the following formulas:

a. NUM = order of the numerator
b. DEN = order of the denominator
c. NUMPN = order of the numerator multiplied by three
d. DENPN = order of the denominator multiplied by three
e. NUMM1 = order of the numerator plus one
f. DENM1 = order of the denominator plus one

These variables must be determined and loaded into memory with the transfer function coefficients before the bilinear transform program can be used.

After equation 5.12 has been performed on the numerator and denominator coefficients, the coefficients that now reside in the memory allocated for the numerator and denominator transfer function coefficients are the coefficients of the difference equation. With the coefficients of the difference equation obtained, a routine

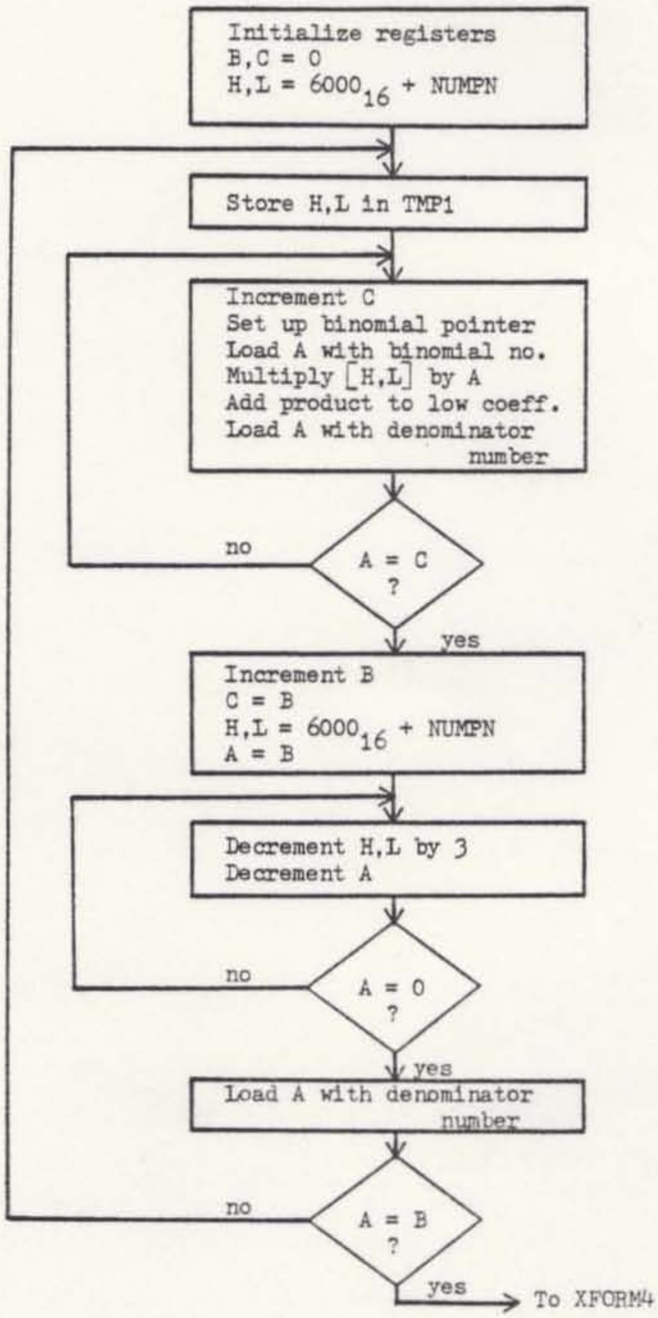

Figure 7. XFORM3 flow chart

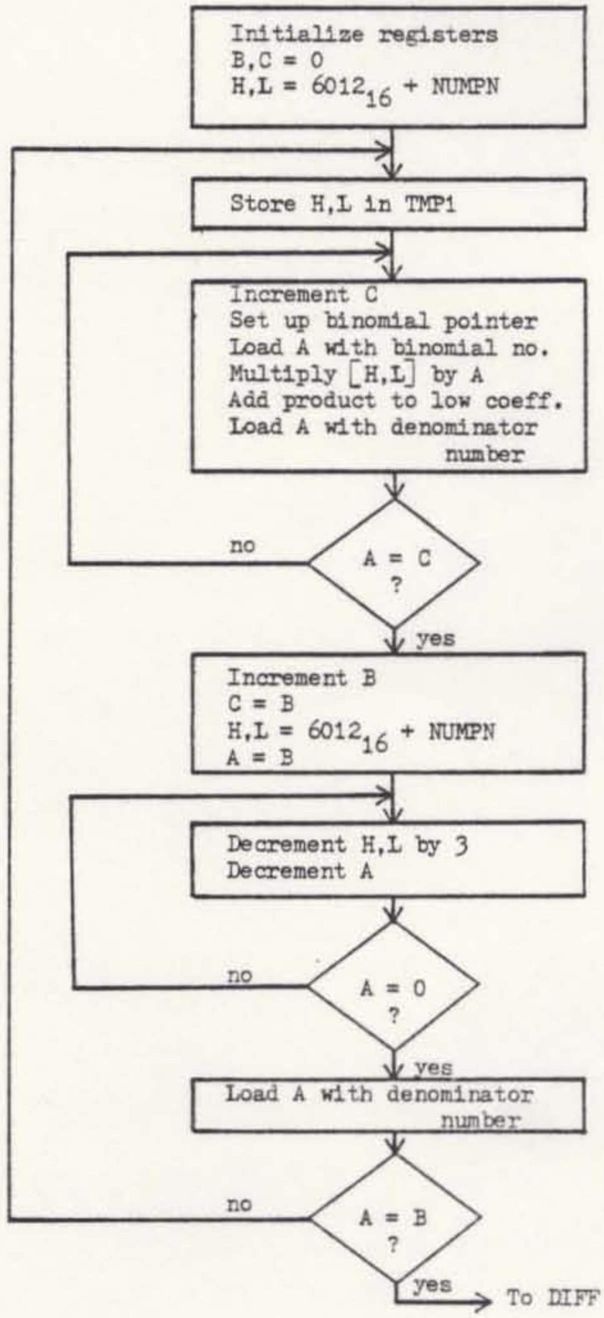

Figure 8. XFORM4 flow chart

must be written to output a string of values based on a string of input values (based on the discrete time mode of the difference equation). Since a system response to a step function is a common method to determine a systems' transient response, a discrete time step function is used as the input string of values. Knowing that the input and output values must be fractional numbers, the input values must be limited in value to prevent the output values from overflowing. Knowing that a realizable transfer functions' output will never exceed twice the input value, an input value limit is chosen to be $\frac{1}{2}$ unit.

Having determined the constraints on the difference equation (equation 2.27), which is transformed into the discrete time domain of equation 2.32, a program can be written to evaluate equation 2.32. Equation 2.32 is restated here as:

$$y(nT) = \frac{\sum_{i=0}^{m} c_{m-i}\, x(nT-iT) - \sum_{i=1}^{m} d_{m-i}\, y(nT-iT)}{d_m} \qquad 5.13)$$

This equation can be broken down into three simpler equations that can be used to design a structured software program. Equation 5.13 can be divided into three subprograms:

a. the summation over the x inputs
b. the summation over the y outputs
c. the division over the entire summation to obtain the present time output.

A flow chart implementing the summation over the inputs is shown in figure 9. This subprogram (DIFF) performs the discrete time coefficient by discrete time input factor multiplication and

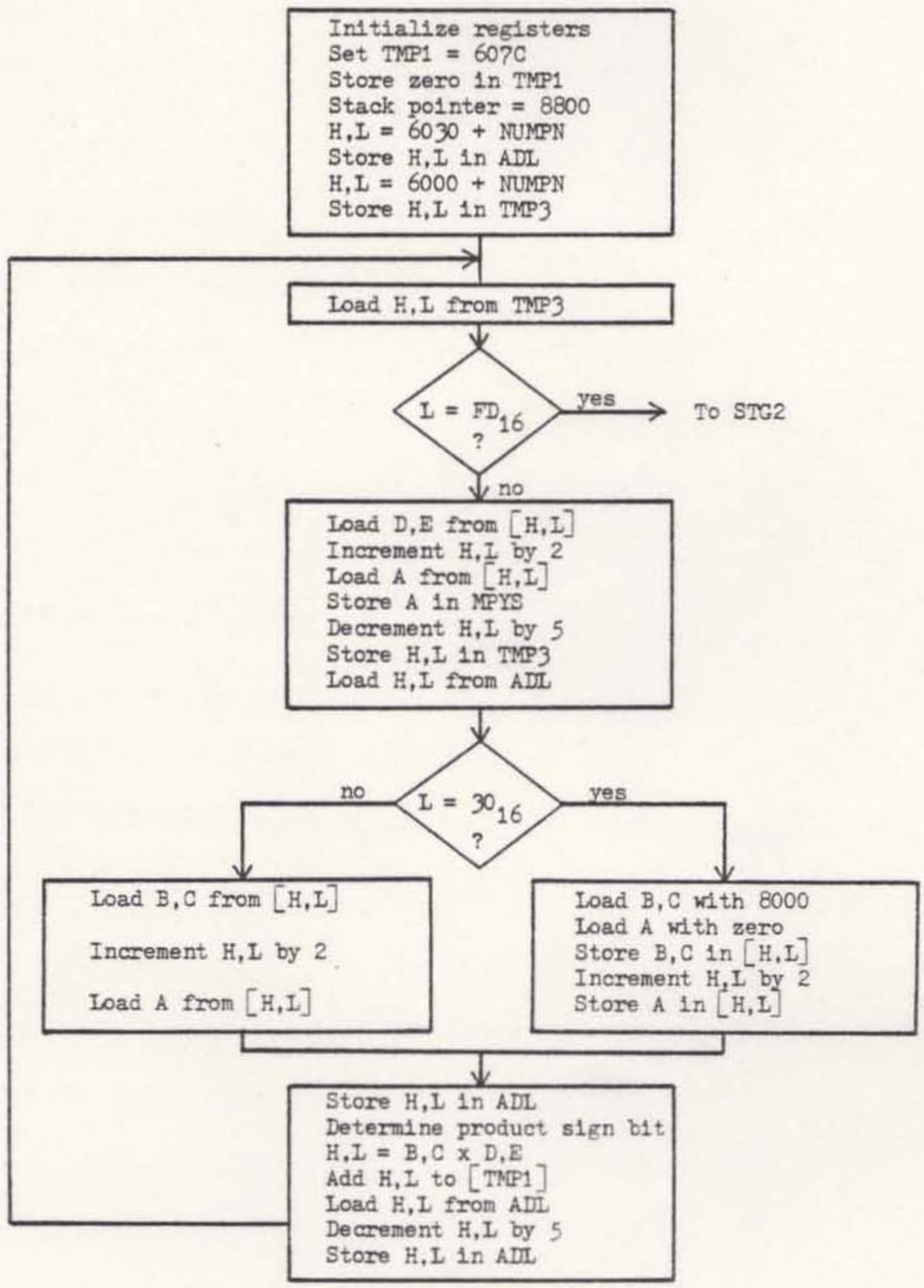

Figure 9. DIFF flow chart

sums these partial products into a memory storage location. For the input at nT, the program inserts an input value of $\frac{1}{2}$ into both the program and the discrete time input factor memory storage location. After the discrete time inputs have all been accounted for in equation 5.13, the discrete time output values must be subtracted from the memory location holding the partial summation over the inputs. This program (STG2) performs the coefficient by discrete time output factor multiplication, performs a twos' compliment on the product and subtracts the product from the overall summation factor. Figure 10 depicts the flow chart for this subprogram. Once all the discrete time output factors have been multiplied and subtracted from the discrete time input factor summation, the present discrete time output, y(nT), must be evaluated. The program (STG3) divides the total summation number by the coefficient d_m to determine the y(nT). The y(nT) is then outputted to an output device for viewing and recording purposes. This subprogram is flow charted in figure 11.

When all the discrete time input and output factors have been evaluated for t=nT, the sampling time point must be incremented to t=nT+T. All the discrete time factors must be shifted back in time by T so that the new sampling time point is nT. Since equation 5.13 is determined from past and present time values for x and y, an increment in time, T, moves all the x and y values back in time by T. Therefore, all the x and y discrete time factors must be shifted in the memory location to match up with their respective position in time. Figure 12 demonstrates how the x and y values are shifted when the sample time point is incremented.

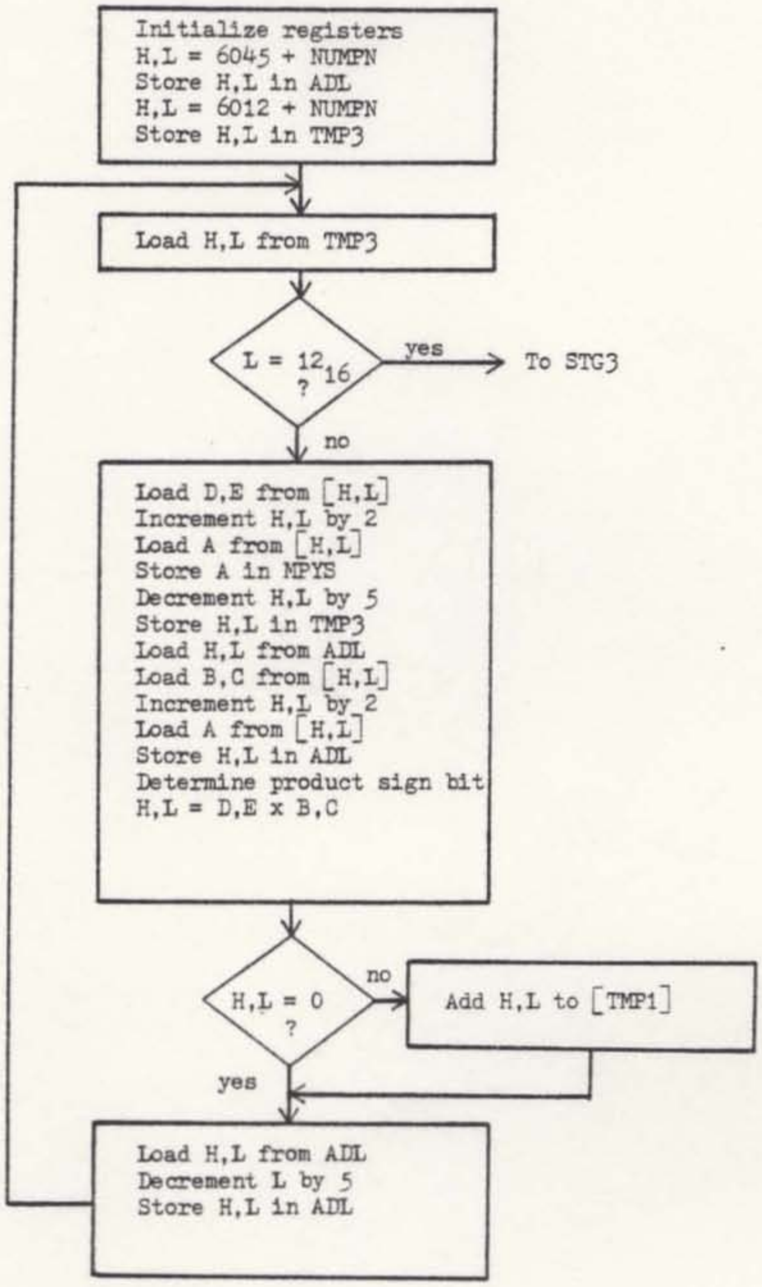

Figure 10. STG2 flow chart

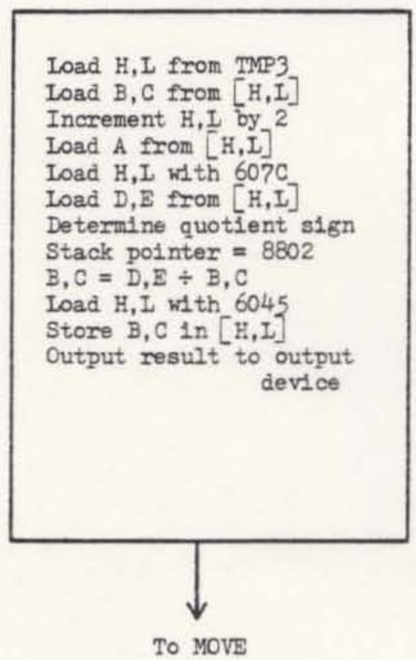

Figure 11. STG3 flow chart

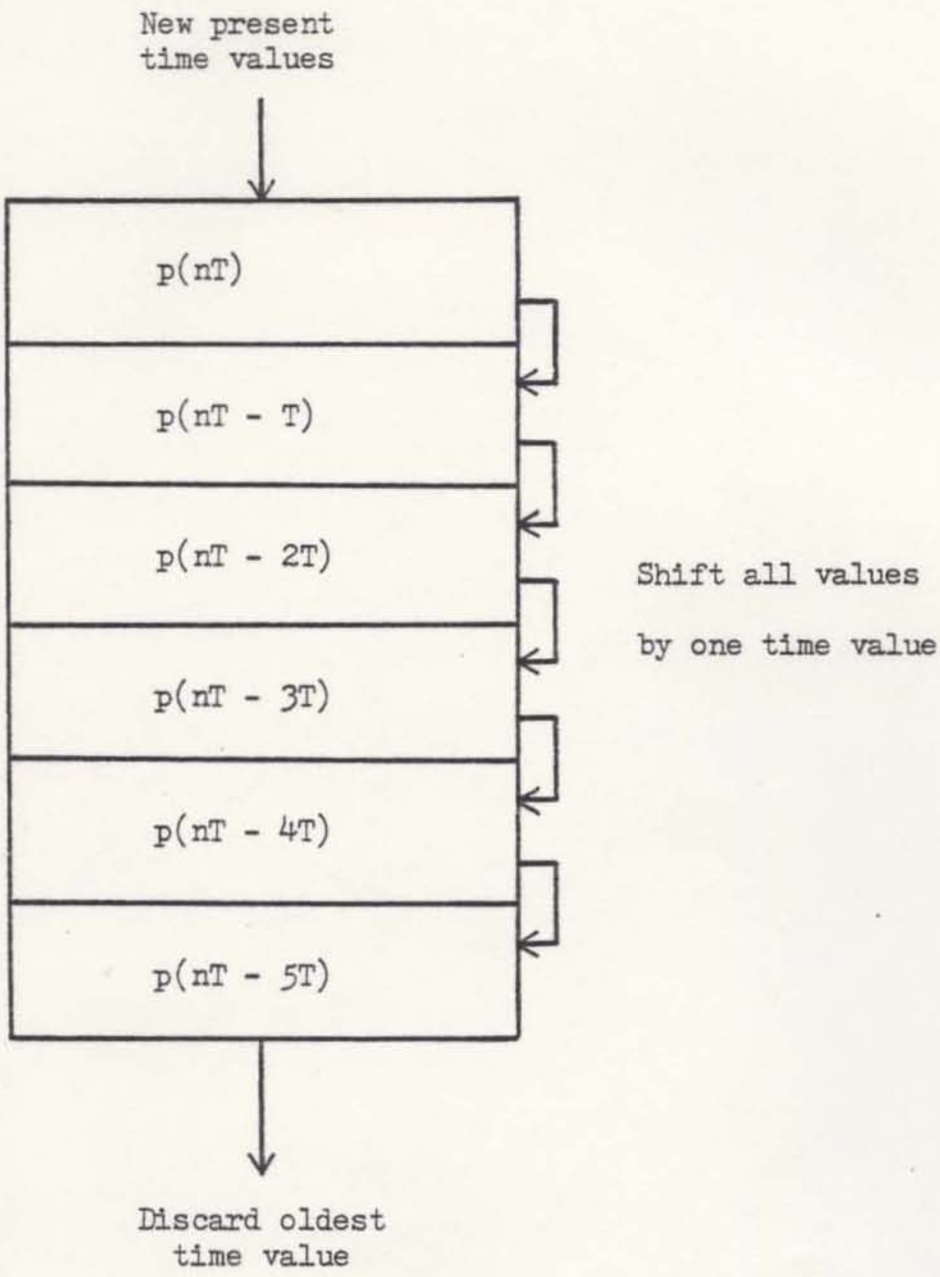

Figure 12. Shifting of difference equation time values

TABLE I

Input - output execution time based on order of transfer function

N	Execution time
1	10,443 usec.
2	16,486 usec.
3	22,529 usec.
4	28,572 usec.
5	34,615 usec.

TABLE II

Maximum omega that input - output

routine can be run in real time

N	$.1\tau$	$.05\tau$	$.01\tau$
1	60.16	30.10	6.01
2	38.11	19.05	3.81
3	27.88	13.94	2.78
4	21.99	10.99	2.19
5	18.15	9.07	1.81

Finally, to ensure that the output string of values occur in a real time mode, any excess execution time must be used up before a new input-output sequence begins. This timing routine must be adjustable based on the updating rate of the output string. The execution time of the input-output routine, based on the order of the transfer function, must be included in the design of the timing routine. Table I displays the execution time of the input-output routine of first to fifth order functions. Based on this information, the maximum frequency that the input-output routine can be operated at, based on the sampling rate, is depicted in Table II.

The end result is a computer program that is capable of performing a bilinear transform algorithm on a differential equation to produce a difference equation. From this difference equation, an output string of discrete time values can be evaluated and produced in a real time mode. By using any transfer function that fits within the constraints of this software program, a real time simulated digital filter can be implemented using this program. With proper interfacing techniques, the software program could actually be used to synthesize a digital filter 'in circuit' in a real time mode.

VI. FILTER IMPLEMENTATION

Having designed a software program to implement the bilinear transform, several filter transfer functions have to be tested on the software program to determine the programs' accuracy. The accuracy of the program can be determined by comparing the output from the bilinear transform program with the output determined from the original transfer function, using the same input conditions. By comparing the two outputs, the sensitivity of the program to data format and scaling parameters can be determined. The performance of the program to known transfer functions will help determine the response from any general transfer function.

Filter designs are based on a set of frequency characteristics that are required for a circuit. Therefore, a filter is a frequency selective device. Normally, a test for a filter would involve implementing a frequency spectrum sweep on the filter and observing the output frequency spectrum. However, a digital filter has a different method to be used to check for accuracy. Based on the original transfer function in s, a continuous time response can be obtained from the analog filter. This continuous time response can then be sampled at intervals of nT (or discretized) to obtain a time sampled response. This response can then be compared to the response from the digital filter, based on the same input, although now discretized. If the digital filter response is accurate, this output should be the same as the discretized response of the analog filter. Upon this basis, the digital filters are tested in the time domain and not in the frequency domain.

Based on the information in Tables I and II, an operating frequency for the test transfer functions is selected to be $\omega = 10$. Sampling rates of $.1\tau$ and $.05\tau$ are used for the output rate of the discrete time equation, based on $\omega = 10$. Three transfer functions are chosen to test the performance of the software program. These transfer functions are:

a. second order low pass
b. third order low pass Butterworth
c. third order low pass Chebyshev with 1 dB ripple

These three transfer functions are sufficient to test the software program, testing different types of transfer functions at different system orders.

The transfer function for the second order low pass filter is:

$$H(s) = \frac{100}{s^2 + 10s + 100} \qquad 6.1)$$

Taking equation 6.1 and allowing $X(s)$ to be a $\frac{1}{2}$ unit step function and then performing an inverse Laplace transform, the resultant transient response is:

$$y(t) = .5 - \frac{1}{\sqrt{3}} e^{-5t} \text{SIN} \left[10\sqrt{3} + \frac{\pi}{3} \right] \quad , \; t \geq 0 \qquad 6.2)$$

From equation 6.1, the scaled transfer functions (using equation 4.9) using $.1\tau$ and $.05\tau$ sampling rates are determined and shown in Table III. Table III displays the coefficients of the transfer function in both decimal and hexadecimal form.

TABLE III

Scaled second order transfer functions

$$H(s) = \frac{.012337}{.125s^2 + .0392699s + .012337}$$

a. $.1\tau$ sampling rate - decimal format

$$H(s) = \frac{.0328}{.2000s^2 + .0A0Ds + .0328}$$

b. $.1\tau$ sampling rate - hexadecimal format

$$H(s) = \frac{.003084}{.125s^2 + .0196349s + .003084}$$

c. $.05\tau$ sampling rate - decimal format

$$H(s) = \frac{.00CA}{.2000s^2 + .0506s + .00CA}$$

d. $.05\tau$ sampling rate - hexadecimal format

The transfer function for the third order Butterworth low pass filter is:

$$H(s) = \frac{1000}{s^3 + 20s^2 + 200s + 1000} \qquad 6.3)$$

Again using a $\frac{1}{2}$ unit step input and taking the inverse Laplace transform, the transient response of equation 6.3 becomes:

$$y(t) = .5 - .5e^{-10t} - \frac{5}{\sqrt{75}} e^{-5t} \text{SIN} [\sqrt{75}\, t] \qquad t \geq 0 \qquad 6.4)$$

Using equation 6.3, the scaled transfer functions using $.1T$ and $.05T$ sampling rates are shown in Table IV, in both decimal and hexadecimal form.

The transfer function for the third order Chebyshev low pass filter with 1 dB ripple is:

$$H(s) = \frac{491.3}{s^3 + 9.8834s^2 + 123.84s + 491.3} \qquad 6.5)$$

Taking equation 6.5 and allowing a $\frac{1}{2}$ unit step input and then taking an inverse Laplace transform, the transient response becomes:

$$y(t) = .5 - .5e^{-4.9417t} - \frac{2.47}{\sqrt{93.314}} e^{-2.471t} \text{SIN} [\sqrt{93.314}\, t] \qquad t \geq 0 \qquad 6.6)$$

From equation 6.5, the scaled transfer functions using $.1T$ and $.05T$ sampling rates are shown in Table V, in both decimal and hexadecimal form.

From equation 6.2, a plot of the response, y(t), versus time is plotted in figure 13. Along with the transient response, the outputs from the discrete time functions are also plotted. Similarly, equation 6.4 and the discrete time function outputs are plotted in

TABLE IV

Scaled third order Butterworth transfer functions

$$H(s) = \frac{.0009689}{.03125s^3 + .0196349s^2 + .0061685s + .0009689}$$

a. $.1\tau$ sampling rate - decimal format

$$H(s) = \frac{.003F}{.0800s^3 + .0506s^2 + .0194s + .003F}$$

b. $.1\tau$ sampling rate - hexadecimal format

$$H(s) = \frac{.0001211}{.03125s^3 + .0098174s^2 + .0015421s + .0001211}$$

c. $.05\tau$ sampling rate - hexadecimal format

$$H(s) = \frac{.0007}{.0800s^3 + .0283s^2 + .0065s + .0007}$$

d. $.05\tau$ sampling rate - hexadecimal format

TABLE V

Scaled third order Chebychev transfer functions

$$H(s) = \frac{.000476}{.03125s^3 + .009703s^2 + .0038195s + .000476}$$

a. $.1\tau$ sampling rate - decimal format

$$H(s) = \frac{.001F}{.0800s^3 + .027Bs^2 + .00FAs + .001F}$$

b. $.1\tau$ sampling rate - hexadecimal format

$$H(s) = \frac{.0000595}{.03125s^3 + .0048515s^2 + .0009548s + .0000595}$$

c. $.05\tau$ sampling rate - decimal format

$$H(s) = \frac{.0003}{.0800s^3 + .013Ds^2 + .003Es + .0003}$$

d. $.05\tau$ sampling rate - hexadecimal format

figure 14, and equation 6.6 and the discrete time function outputs are plotted in figure 15.

Returning to figure 13, the outputs from the $.1\tau$ and $.05\tau$ discrete time functions are seen to closely follow the transient response. The output from the $.05\tau$ discrete time function 'tracks' the transient response more accurately than the $.1\tau$ discrete time function, due to more samples per time period. The steady state value for the transient response is $.8000_{16}$ ($.5000_{10}$) and the steady state value for the $.1\tau$ discrete time function is $.800B_{16}$ ($.5001678_{10}$) and for the $.05\tau$ discrete time function is $.7FD7_{16}$ ($.4993743_{10}$). In both cases, the steady state error is less than .125% for $.7FD7_{16}$ and less than .033% for $.800B_{16}$. Both cases represent very close approximation to the transient response.

Figure 14 shows that the $.1\tau$ discrete time function accurately follows the transient response, while the $.05\tau$ discrete time function does not match the transient response characteristics. Both discrete time functions settle down to a steady state value, with the $.1\tau$ discrete time function having a $.7EC9_{16}$ ($.495254_{10}$) value and the $.05\tau$ discrete time function having a $.74BF_{16}$ ($.456039_{10}$) value. The transient response has a steady state value of $.8000_{16}$ ($.5000_{10}$). These steady state values represent a steady state error of .949% for $.7EC9_{16}$ and 8.792% for $.74BF_{16}$.

In figure 15, the $.1\tau$ discrete time function accurately follows the transient response, while the $.05\tau$ discrete time function does not match the transient response characteristics at all. The steady state value of the $.1\tau$ discrete time function is $.7D10_{16}$

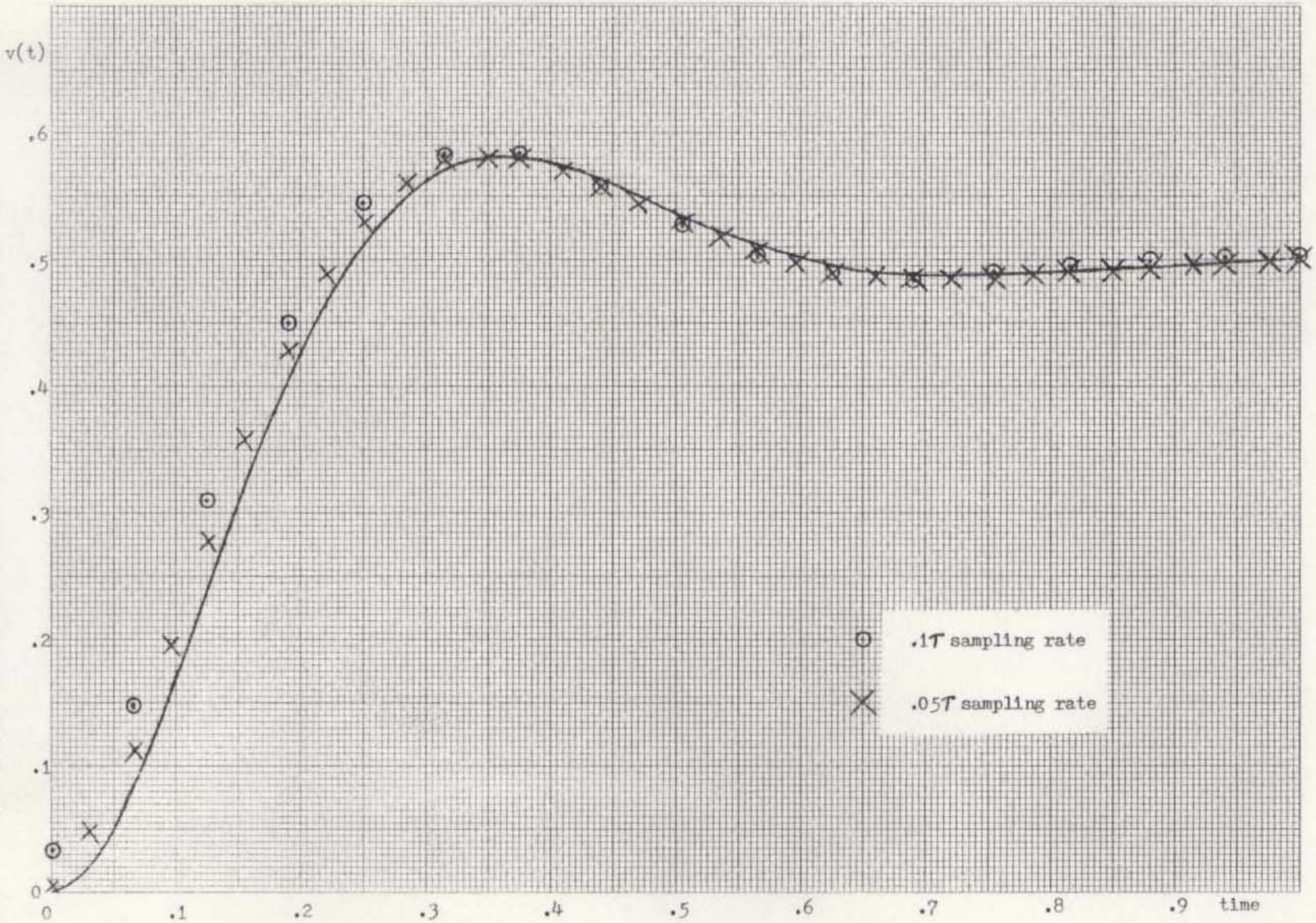

Figure 13. Second Order Transfer Function

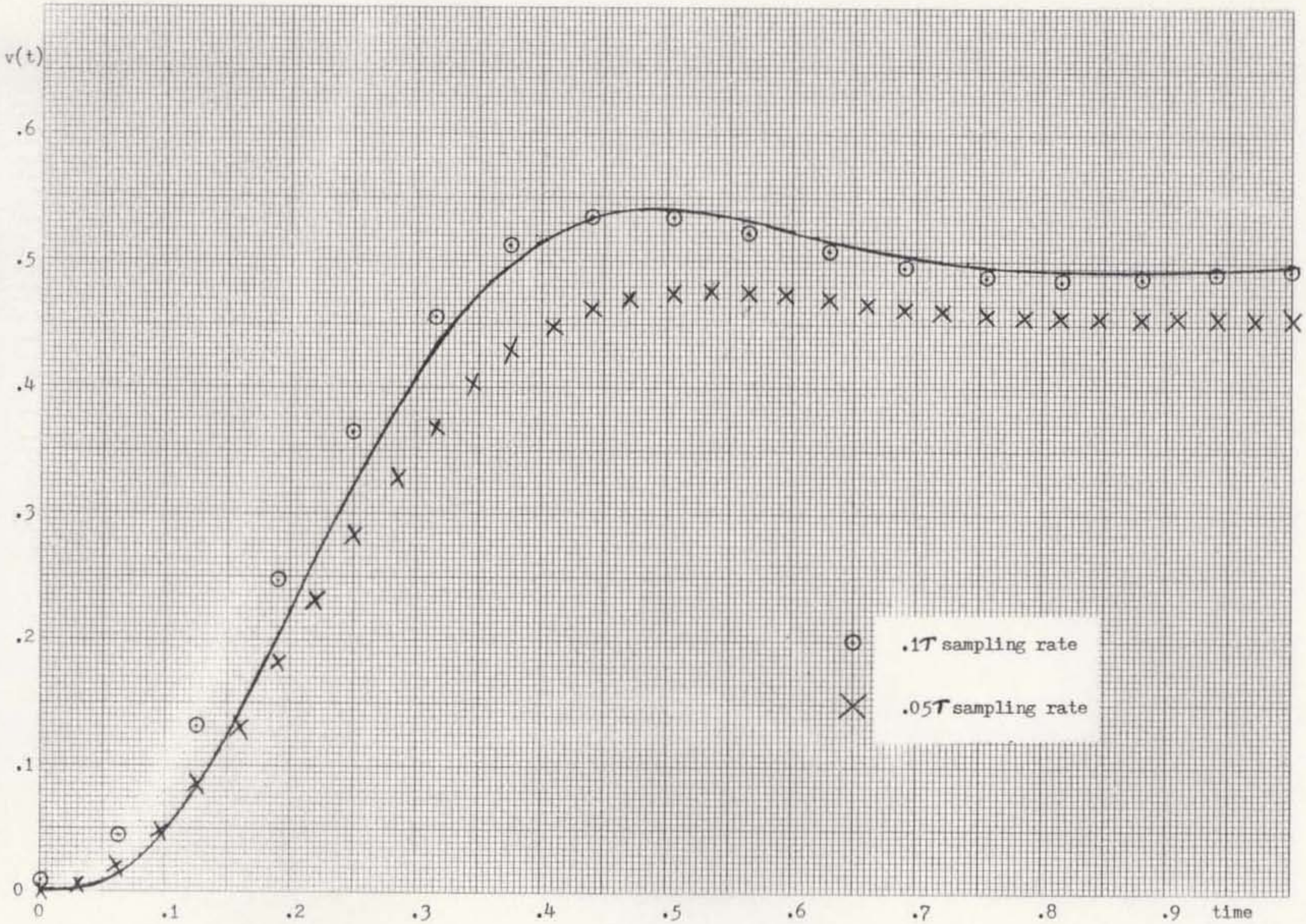

Figure 14. Third Order Butterworth Transfer Function

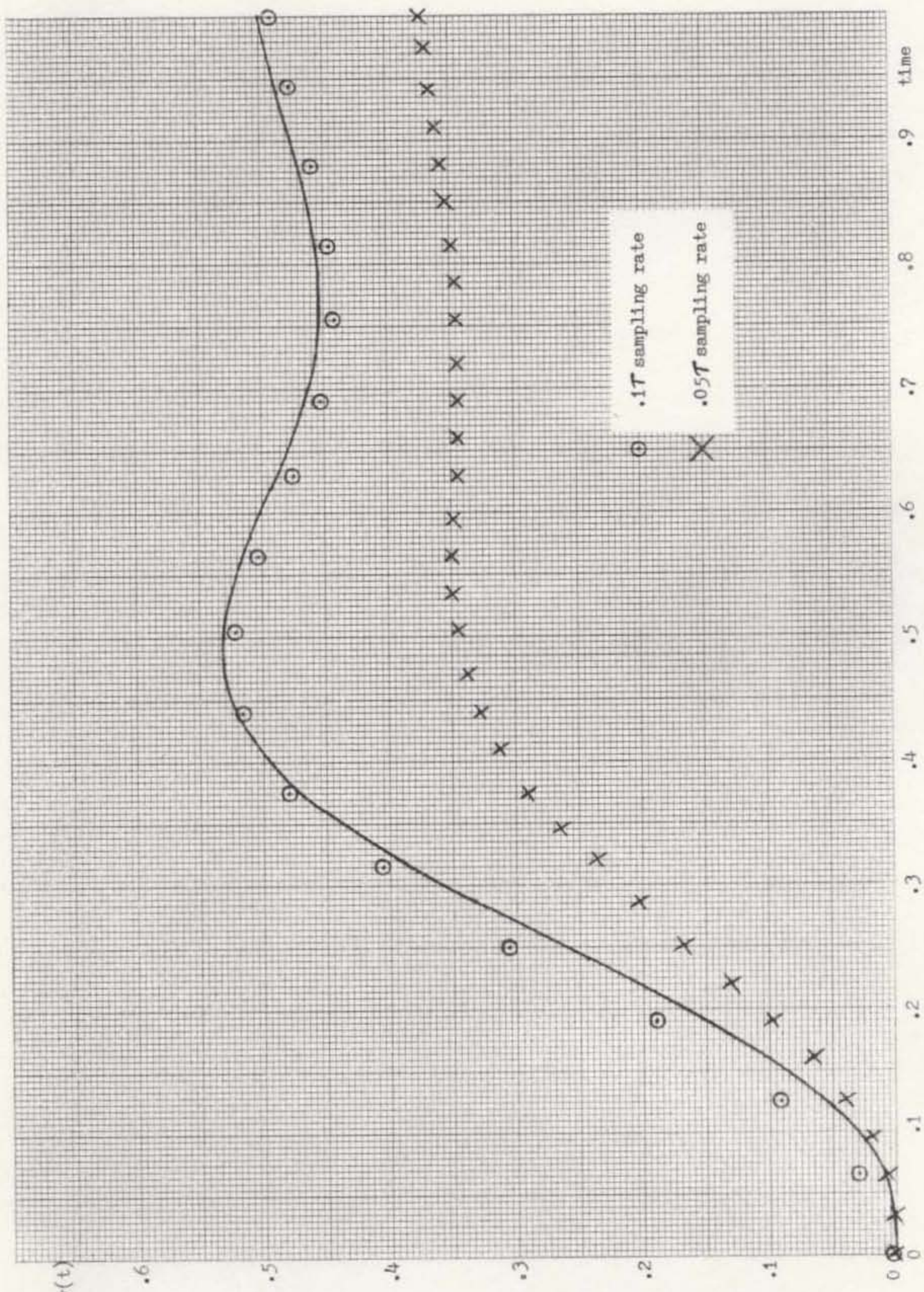

Figure 15. Third Order Chebychev Transfer Function

($.488525_{10}$) and for the $.05T$ discrete time function is $.60EB_{16}$ ($.378585_{10}$). The transient response has a steady state error of 2.295% for $.7D10_{16}$ and 24.283% for $.60EB_{16}$.

In all the discrete time functions, there exists a larger steady state error for the $.05T$ sampling rate than for the $.1T$ sampling rate. An examination of the scaled transfer functions for $.1T$ and $.05T$ sampling rates in hexadecimal format (parts b and d in Tables III, IV, V) show that the number of non-zero bits in the coefficient drops by as much as three bits as the sampling rate increases from $.1T$ to $.05T$. For the higher order systems, this leaves only two or three non-zero bits for the zero order coefficients. The result of the coefficients being rounded off and expressed in such small, truncated numbers is that these coefficients produce round off errors when shifted, added and multiplied by the bilinear transform. As long as there are sufficient bits in the coefficients to retain data accuracy, the bilinear transform closely matches the transient response (as in the $.1T$ case, all transfer functions). Once the data accuracy is lost, due to insufficient bits, the bilinear transform is using truncated data words, and the output from the discrete time equation is a poor approximation of the transient response. The result is a tradeoff between sampling rate and data accuracy; data accuracy diminishes with higher sampling rates and the output is inaccurate. With a low sampling rate, the output has less than 1% error for two of the functions tested at $.1T$ and less than 2.3% error for the Chebychev function at $.1T$.

VII. RESULTS AND CONCLUSIONS

The software program that is implemented in this thesis is basically two separate programs linked together. One program performs a bilinear transform on a transfer function to generate coefficients for a difference equation. The other program actually performs an input-output operation on the coefficients of the difference equation. Through the implementation of both of these software modules as one larger program, the performance of these programs can be evaluated. By evaluating these performance characteristics, the benefits/disadvantages of the programs are revealed.

The software program written to implement the bilinear transform algorithm was designed around the need to calculate the data as quickly as possible. To help increase calculation speed, a fixed point notation was used to represent the data. Double precision notation was needed to insure adequate word length during the calculations. To insure that the data was represented accurately, the data was scaled to a fractional form. To reduce calculation time on the bilinear transform equations, a scaling factor was designed to frequency scale the transfer function to a standardized frequency, based on the sampling rate of the discrete time equation. All these techniques were used in writing the bilinear transform algorithm software program.

Based on the results of filter transform functions implemented in Chapter VI, the bilinear transform software program results in an output error less than 2.3% when the data is accurately represented (minimum of last 7 of 16 bits are non zero or contain data information). When the 16 bit data word truncates the value of the real coefficients, the bilinear transform can provide an output error greater than 8% of the real transient response. The truncation of data occurs when the sampling rate is increased, causing the scaling factor to decrease the values of the transfer functions' coefficients. From this knowledge, there are several solutions to retain data accuracy with increasing sampling speed. Among these ideas include:

a. using a 16 bit microprocessor with double precision (32 bit) word length
b. using a different scaling technique
c. using a floating point notation
d. developing new equations to implement the bilinear transform.

Using a 32 bit word would increase the data accuracy, until high sampling rates are needed, where the data would again be truncated. A different scaling technique could imply rewriting the algorithms, possibly slowing down execution time. Floating point notation would allow a wide range of data values, but would slow down execution time. Other new algorithm equations are not yet developed to execute the bilinear transform with minimum memory. There appears to be no single best solution to this problem. Using any alternate approach that will not drastically increase execution time can be considered a feasible solution.

From the information supplied in Tables I and II, the maximum operating frequency of the program is limited by the input-output routine. An analysis of the input-output program reveals that a major amount of execution time is spent in software multiply and divide routines. The data acquisition and add routines are presently using minimal execution time based on the 16 bit data word. An improvement in this program would be the implementation of a hardware or firmware multiply/divide routine to decrease the execution time. By decreasing execution time, the maximum frequency obtainable is increased. Since the input-output routine is a very straightforward process, the algorithms need not be modified. The execution time can be reduced by using hardware or firmware multiply and divide routines.

The algorithms designed to perform an 'in place' operation, based on the bilinear transform, can result in output errors less than 2.3% on a microprocessor system. Based on sampled outputs from the bilinear transform program versus outputs from the original transfer function, the program data matches the theoretical data within a 2.3% error provided that the last 7 bits of the 16 bit data word contain data information. Faster input-output operations can be obtained by substituting a hardware or firmware multiply/divide routine for the present software routine. With these modifications, a sufficiently powerful real-time digital filter can be designed around a small memory microprocessor, with continuously programmable features that make this system extremely attractive for digital filter design implementation. Furthermore, the generalized procedure for the second order accuracy bilinear approach implies a search for similar higher order accuracy algorithms that could be beneficial to state of the art digital filter design.

APPENDIX A

INTEL 8080 Assembly Program Listing

```
ASM80 :F1:FILTER.SRC SYMBOLS XREF


ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        XFORM    PAGE    1
S TO Z TRANSFORM ALGORITHMS

 LOC  OBJ        SEQ         SOURCE STATEMENT

                   1 $ TITLE ('S TO Z TRANSFORM ALGORITHMS')
                   2 NAME XFORM
 6024              3 TMP1 EQU 6024H
 6026              4 TMP2 EQU 6026H
 6027              5 MPYL EQU 6027H
 6028              6 MPYM EQU 6028H
 6029              7 MPYS EQU 6029H
 605A              8 NUM EQU 605AH
 605B              9 DEN EQU 605BH
 605C             10 NUMPN EQU 605CH
 605D             11 DENPN EQU 605DH
 605E             12 NUMM1 EQU 605EH
 605F             13 DENM1 EQU 605FH
                  14
 6000             15 ORG 6000H
 0060             16 TABLE: DS 60H         ; RESERVE 96 BYTES
 6060 01          17 BINOM: DB 1,1,2,3,4,5 ; BINOMIAL LOOKUP TABLE
 6061 01
 6062 02
 6063 03
 6064 04
 6065 05
 6066 03          18 DB 3,3,6,6,10,10
 6067 03
 6068 06
 6069 06
 606A 0A
 606B 0A
 606C 04          19 DB 4,4,4,10,10,10
 606D 04
 606E 04
 606F 0A
 6070 0A
 6071 0A
 6072 0A          20 DB 10,10,5
 6073 0A
 6074 05
 000B             21 DS 0BH                ; RESERVE 11 BYTES
                  22
 6080 010000      23 XFORM: LXI B,00H      ; SET B & C TO ZERO
 6083 210060      24 LXI H,TABLE           ; SET H,L = 6000H
 6086 222460      25 CNTR: SHLD TMP1       ; STORE H,L IN TMP1
 6089 116060      26 CONT: LXI D,BINOM     ; D,E GETS 6060H
 608C 0C          27 INR C
 608D CDEA61      28 CALL MULT             ; A GETS B X C
 6090 83          29 ADD E                 ; SET BINOMIAL POINTER
 6091 5F          30 MOV E,A
 6092 2C          31 INR L
 6093 2C          32 INR L
 6094 2C          33 INR L
 6095 1A          34 LDAX D                ; A GETS BINOMIAL NUMBER
 6096 CD5362      35 CALL MPY              ; MULTIPLY BIN. # BY COEFFICIENT
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0      XFORM   PAGE   2
S TO Z TRANSFORM ALGORITHMS

 LOC  OBJ         SEQ        SOURCE STATEMENT

 6099 CD0362       36 CALL ADM              ; ADD MPY # TO LOW COEFFICIENT
 609C 3A5A60       37 LDA NUM               ; CHECK FOR HIGHEST COEFFICIENT
 609F B9           38 CMP C
 60A0 C28960       39 JNZ CONT              ; LOOP BACK IF NOT DONE
 60A3 04           40 INR B
 60A4 48           41 MOV C,B               ; C GETS B
 60A5 210060       42 LXI H,TABLE           ; H, L GETS 6000H
 60A8 78           43 MOV A,B               ; A GETS B
 60A9 2C           44 INCR: INR L           ; SET UP NEW COEFFICIENT POINTER
 60AA 2C           45 INR L
 60AB 2C           46 INR L
 60AC 3D           47 DCR A                 ; A GETS A - 1
 60AD C2A960       48 JNZ INCR              ; UPDATE COEFFICIENT POINTER
 60B0 3A5A60       49 LDA NUM               ; A GETS NUMERATOR ORDER
 60B3 B8           50 CMP B
 60B4 C28660       51 JNZ CNTR              ; LOOP BACK IF ALL COEFFICIENTS
                   52                       ; NOT DONE
 60B7 010000       53 XFORM2: LXI B,00H     ; B, C SET TO ZERO
 60BA 211260       54 LXI H,TABLE+12H       ; H, L GETS 6012H
 60BD 222460       55 CNTR2: SHLD TMP1      ; STORE H, L IN TMP1
 60C0 116060       56 CONT2: LXI D,BINOM    ; D, E GETS 6060H
 60C3 0C           57 INR C
 60C4 CDEA61       58 CALL MULT             ; A GETS B X C
 60C7 83           59 ADD E                 ; SET BINOMIAL POINTER
 60C8 5F           60 MOV E,A
 60C9 2C           61 INR L
 60CA 2C           62 INR L
 60CB 2C           63 INR L
 60CC 1A           64 LDAX D                ; A GETS BINOMIAL NUMBER
 60CD CD5362       65 CALL MPY              ; MULTIPLY BIN. # BY COEFFICIENT
 60D0 CD0362       66 CALL ADM              ; ADD MPY # TO LOW COEFFICIENT
 60D3 3A5B60       67 LDA DEN               ; CHECK FOR HIGHEST COEFFICIENT
 60D6 B9           68 CMP C
 60D7 C2C060       69 JNZ CONT2             ; LOOP BACK IF NOT DONE
 60DA 04           70 INR B
 60DB 48           71 MOV C,B               ; C GETS B
 60DC 211260       72 LXI H,TABLE+12H       ; H,L GETS 6012H
 60DF 78           73 MOV A,B
 60E0 2C           74 INCR2: INR L          ; SET UP NEW COEFFICIENT POINTER
 60E1 2C           75 INR L
 60E2 2C           76 INR L
 60E3 3D           77 DCR A
 60E4 C2E060       78 JNZ INCR2             ; UPDATE COEFFICIENT POINTER
 60E7 3A5B60       79 LDA DEN               ; A GETS DENOMINATOR ORDER
 60EA B8           80 CMP B
 60EB C2BD60       81 JNZ CNTR2             ; LOOP BACK IF ALL COEFFICIENTS
                   82                       ; NOT DONE
 60EE 0601         83 X2NA: MVI B,1         ; B GETS 1
 60F0 3A5A60       84 LDA NUM               ; A GETS NUMERATOR ORDER
 60F3 B8           85 CMP B
 60F4 CA2E61       86 JZ X2NB               ; BREAK OUT IF NO COEFFICIENT AFFECTED
 60F7 210360       87 LXI H,TABLE+3         ; H,L GETS 6003H
 60FA 0E00         88 RPT: MVI C,0
 60FC 56           89 MOV D,M               ; LOAD COEFFICIENT INTO D,E
 60FD 2C           90 INR L
```

```
 LOC  OBJ         SEQ        SOURCE STATEMENT

 60FE 5E           91 MOV E,M
 60FF 2C           92 INR L
 6100 7E           93 MOV A,M              ; LOAD COEFFICIENT SIGN IN A
 6101 322660       94 STA TMP2             ; STORE A IN TMP2
 6104 37           95 SHFT: STC            ; SET CARRY BIT
 6105 3F           96 CMC                  ; CLEAR CARRY BIT
 6106 7B           97 MOV A,E              ; MULTIPLY COEFFICIENT BY 2
 6107 17           98 RAL
 6108 5F           99 MOV E,A
 6109 7A          100 MOV A,D
 610A 17          101 RAL
 610B 57          102 MOV D,A
 610C 3A2660      103 LDA TMP2             ; COMPLIMENT SIGN BIT
 610F 2F          104 CMA
 6110 322660      105 STA TMP2
 6113 0C          106 INR C
 6114 78          107 MOV A,B              ; A GETS B
 6115 B9          108 CMP C
 6116 C20461      109 JNZ SHFT             ; CONTINUE MULTIPLY IF NOT DONE
 6119 3A2660      110 LDA TMP2             ; CLEAN UP SIGN BIT
 611C E680        111 ANI 80H
 611E 77          112 MOV M,A              ; RESTORE COEFFICIENT IN MEMORY
 611F 2D          113 DCR L
 6120 73          114 MOV M,E
 6121 2D          115 DCR L
 6122 72          116 MOV M,D
 6123 2C          117 INR L                ; INCREMENT POINTER TO NEW COEFFICIENT
 6124 2C          118 INR L
 6125 2C          119 INR L
 6126 04          120 INR B
 6127 3A5E60      121 LDA NUMM1            ; A GETS TOP OF STORAGE STACK NUMBER
 612A B8          122 CMP B
 612B C2FA60      123 JNZ RPT              ; REPEAT SHIFT IF NEW COEFFICIENT AVAILABLE
                  124
 612E 0601        125 X2NB: MVI B,1        ; B GETS 1
 6130 211560      126 LXI H,TABLE+15H      ; H,L GETS 6015H
 6133 0E00        127 RPT2: MVI C,0
 6135 56          128 MOV D,M              ; LOAD COEFFICIENT INTO D,E
 6136 2C          129 INR L
 6137 5E          130 MOV E,M
 6138 2C          131 INR L
 6139 7E          132 MOV A,M
 613A 322660      133 STA TMP2             ; STORE SIGN BIT IN TMP2
 613D 37          134 SHFT2: STC           ; SET CARRY
 613E 3F          135 CMC                  ; CLEAR CARRY
 613F 7B          136 MOV A,E              ; MULTIPLY COEFFICIENT BY 2
 6140 17          137 RAL
 6141 5F          138 MOV E,A
 6142 7A          139 MOV A,D
 6143 17          140 RAL
 6144 57          141 MOV D,A
 6145 3A2660      142 LDA TMP2             ; COMPLIMENT SIGN BIT
 6148 2F          143 CMA
 6149 322660      144 STA TMP2
 614C 0C          145 INR C
```

```
LOC  OBJ         SEQ       SOURCE STATEMENT

614D 78           146 MOV A,B
614E B9           147 CMP C
614F C23D61       148 JNZ SHFT2              ; CONTINUE MULTIPLY IF NOT DONE
6152 3A2660       149 LDA TMP2               ; CLEAN UP SIGN BIT
6155 E680         150 ANI 80H
6157 77           151 MOV M,A                ; RESTORE COEFFICIENT TO MEMORY
6158 2D           152 DCR L
6159 73           153 MOV M,E
615A 2D           154 DCR L
615B 72           155 MOV M,D
615C 2C           156 INR L                  ; INCREMENT POINTER TO NEW COEFFICIENT
615D 2C           157 INR L
615E 2C           158 INR L
615F 04           159 INR B
6160 3A5F60       160 LDA DENM1              ; A GETS TOP OF STORAGE STACK NUMBER
6163 B8           161 CMP B
6164 C23361       162 JNZ RPT2               ; REPEAT SHIFT IF NEW COEFFICIENT AVAILABLE
                  163
6167 010000       164 XFORM3: LXI B,00H      ; B & C SET TO ZERO
616A 210060       165 LXI H,TABLE            ; SET H,L TO POINT TO LOW COEFFICIENT
616D 3A5C60       166 LDA NUMPN
6170 85           167 ADD L
6171 6F           168 MOV L,A
6172 222460       169 CNTR3: SHLD TMP1       ; STORE H,L IN TMP1
6175 116060       170 CONT3: LXI D,BINOM     ; D,E GETS 6060H
6178 0C           171 INR C
6179 CDEA61       172 CALL MULT              ; A GETS B X C
617C 83           173 ADD E                  ; SET BINOMIAL POINTER
617D 5F           174 MOV E,A
617E 2D           175 DCR L
617F 2D           176 DCR L
6180 2D           177 DCR L
6181 1A           178 LDAX D                 ; A GETS BINOMIAL NUMBER
6182 CD5362       179 CALL MPY               ; MULTIPLY BIN. # BY COEFFICIENT
6185 CD0362       180 CALL ADM               ; ADD MPY. # TO LOW COEFFICIENT
6188 3A5B60       181 LDA DEN                ; CHECK FOR HIGHEST COEFFICIENT
618B B9           182 CMP C
618C C27561       183 JNZ CONT3              ; LOOP BACK IF NOT DONE
618F 04           184 INR B
6190 48           185 MOV C,B                ; C GETS B
6191 210060       186 LXI H,TABLE            ; H,L GETS 6000H
6194 3A5C60       187 LDA NUMPN              ; SET H,L TO POINT AT NEW COEFFICIENT
6197 85           188 ADD L
6198 6F           189 MOV L,A
6199 78           190 MOV A,B
619A 2D           191 INCR3: DCR L
619B 2D           192 DCR L
619C 2D           193 DCR L
619D 3D           194 DCR A
619E C29A61       195 JNZ INCR3              ; UPDATE COEFFICIENT POINTER
61A1 3A5B60       196 LDA DEN                ; A GETS DENOMINATOR ORDER
61A4 B8           197 CMP B
61A5 C27261       198 JNZ CNTR3              ; LOOP BACK IF ALL COEFFICIENTS
                  199                        ; NOT DONE
61A8 010000       200 XFORM4: LXI B,00H      ; SET B & C TO ZERO
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        XFORM   PAGE   5
S TO Z TRANSFORM ALGORITHMS

 LOC  OBJ         SEQ       SOURCE STATEMENT

 61AB 211260      201 LXI H,TABLE+12H      ; H,L GETS 6012H
 61AE 3A5D60      202 LDA DENPN            ; SET H,L TO POINT AT LOW COEFFICIENT
 61B1 85          203 ADD L
 61B2 6F          204 MOV L,A
 61B3 222460      205 CNTR4: SHLD TMP1     ; STORE H,L IN TMP1
 61B6 116060      206 CONT4: LXI D,BINOM   ; D,E GETS 6060H
 61B9 0C          207 INR C
 61BA CDEA61      208 CALL MULT            ; A GETS B X C
 61BD 83          209 ADD E                ; SET BINOMIAL POINTER
 61BE 5F          210 MOV E,A
 61BF 2D          211 DCR L
 61C0 2D          212 DCR L
 61C1 2D          213 DCR L
 61C2 1A          214 LDAX D               ; A GETS BINOMIAL NUMBER
 61C3 CD5362      215 CALL MPY             ; MPY BIN. # BY COEFFICIENT
 61C6 CD0362      216 CALL ADM             ; ADD MPY # TO LOW COEFFICIENT
 61C9 3A5B60      217 LDA DEN              ; CHECK FOR HIGHEST COEFFICIENT
 61CC B9          218 CMP C
 61CD C2B661      219 JNZ CONT4            ; LOOP BACK IF NOT DONE
 61D0 04          220 INR B
 61D1 48          221 MOV C,B              ; C GETS B
 61D2 211260      222 LXI H,TABLE+12H      ; H,L GETS 6012H
 61D5 3A5D60      223 LDA DENPN            ; SET H,L TO POINT AT NEW COEFFICIENT
 61D8 85          224 ADD L
 61D9 6F          225 MOV L,A
 61DA 78          226 MOV A,B
 61DB 2D          227 INCR4: DCR L
 61DC 2D          228 DCR L
 61DD 2D          229 DCR L
 61DE 3D          230 DCR A
 61DF C2DB61      231 JNZ INCR4            ; UPDATE COEFFICIENT POINTER
 61E2 3A5B60      232 LDA DEN              ; A GETS DENOMINATOR ORDER
 61E5 B8          233 CMP B
 61E6 C2B361      234 JNZ CNTR4            ; LOOP BACK IF ALL COEFFICIENTS NOT DONE
                  235
 61E9 76          236 HLT                  ; END OF PROGRAM
                  237                      ; MULTIPLY ROUTINE FOR BINOMIAL POINTER
 61EA C5          238 MULT: PUSH B         ; PUSH REGISTERS ON STACK
 61EB D5          239 PUSH D
 61EC E5          240 PUSH H
 61ED 78          241 MOV A,B              ; A GETS B
 61EE FE00        242 CPI 0
 61F0 CAFE61      243 JZ ONE               ; TEST FOR B = 0
 61F3 3E00        244 MVI A,0              ; A GETS 0
 61F5 80          245 MLTY: ADD B          ; ADD B TO A C TIMES
 61F6 0D          246 DCR C
 61F7 C2F561      247 JNZ MLTY             ; CHECK IF C = 0
 61FA E1          248 FINAL: POP H         ; POP REGISTERS OFF STACK
 61FB D1          249 POP D
 61FC C1          250 POP B
 61FD C9          251 RET                  ; END SUBROUTINE
 61FE 3E01        252 ONE: MVI A,1         ; A GETS 1
 6200 C3FA61      253 JMP FINAL
                  254                      ; ADD ROUTINE
 6203 E5          255 ADM: PUSH H          ; PUSH ON STACK
```

```
 LOC  OBJ         SEQ       SOURCE STATEMENT

 6204 D5          256 PUSH D
 6205 C5          257 PUSH B
 6206 F5          258 PUSH PSW
 6207 2A2460      259 LHLD TMP1            ; H,L GET LOW COEFFICIENT ADDRESS
 620A 56          260 MOV D,M              ; D,E GET LOW COEFFICIENT
 620B 23          261 INX H
 620C 5E          262 MOV E,M
 620D 23          263 INX H
 620E 7E          264 MOV A,M              ; A GETS SIGN BIT
 620F 47          265 MOV B,A
 6210 FE80        266 CPI 80H              ; 2'S COMPLIMENT IF NEGATIVE
 6212 CA3D62      267 JZ COMP
 6215 2A2760      268 SET1: LHLD MPYL      ; H,L GET MPY NUMBER
 6218 3A2960      269 LDA MPYS             ; A GETS SIGN BIT
 621B 4F          270 MOV C,A
 621C FE80        271 CPI 80H              ; 2'S COMPLIMENT IF NEGATIVE
 621E CA4562      272 JZ COMP2
 6221 78          273 SET2: MOV A,B        ; CHECK SIGNS OF BOTH NUMBERS
 6222 A9          274 XRA C
 6223 47          275 MOV B,A
 6224 3E00        276 MVI A,0
 6226 19          277 DAD D                ; ADD D,E TO H,L
 6227 1F          278 RAR                  ; PUT SIGN BIT IN A
 6228 A8          279 XRA B                ; FIGURE SIGN OF SUM
 6229 17          280 RAL
 622A DA4B62      281 JC COMP3             ; 2'S COMPLIMENT IF NEGATIVE
 622D 3E00        282 MVI A,0
 622F EB          283 INSRT: XCHG          ; SWAP D,E WITH H,L
 6230 2A2460      284 LHLD TMP1            ; H,L GET LOW COEFFICIENT ADDRESS
 6233 72          285 MOV M,D              ; STORE COEFFICIENT IN MEMORY
 6234 23          286 INX H
 6235 73          287 MOV M,E
 6236 23          288 INX H
 6237 77          289 MOV M,A
 6238 F1          290 POP PSW              ; POP REGISTERS OFF STACK
 6239 C1          291 POP B
 623A D1          292 POP D
 623B E1          293 POP H
 623C C9          294 RET                  ; END SUBROUTINE
                  295
 623D EB          296 COMP: XCHG           ; SWAP D,E WITH H,L
 623E CD6F62      297 CALL MOD1            ; 2'S COMPLIMENT ROUTINE
 6241 EB          298 XCHG                 ; SWAP D,E WITH H,L
 6242 C31562      299 JMP SET1             ; GO TO SET1
                  300
 6245 CD6F62      301 COMP2: CALL MOD1
 6248 C32162      302 JMP SET2
 624B CD6F62      303 COMP3: CALL MOD1
 624E 3E80        304 MVI A,80H
 6250 C32F62      305 JMP INSRT
                  306                      ; MULTIPLY ROUTINE
 6253 C5          307 MPY: PUSH B          ; PUSH REGISTERS ON STACK
 6254 D5          308 PUSH D
 6255 E5          309 PUSH H
 6256 F5          310 PUSH PSW
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        XFORM    PAGE    7
S TO Z TRANSFORM ALGORITHMS

  LOC  OBJ         SEQ        SOURCE STATEMENT

  6257 56          311 MOV D,M              ; D,E GET COEFFICIENT
  6258 23          312 INX H
  6259 5E          313 MOV E,M
  625A 23          314 INX H
  625B 7E          315 MOV A,M
  625C 322960      316 STA MPYS             ; SIGN BIT STORED IN MPY5
  625F F1          317 POP PSW
  6260 210000      318 LXI H,00H            ; H,L = 0
  6263 19          319 MPY2: DAD D          ; ADD D,E TO H,L
  6264 3D          320 DCR A
  6265 C26362      321 JNZ MPY2             ; CONTINUE TO ADD IF A ≠ 0
  6268 222760      322 SHLD MPYL            ; H,L STORED IN MPYL
  626B E1          323 POP H                ; POP REGISTERS OFF STACK
  626C D1          324 POP D
  626D C1          325 POP B
  626E C9          326 RET                  ; END SUBROUTINE
                   327
  626F D5          328 MOD1: PUSH D         ; PUSH D ON STACK
  6270 7C          329 MOV A,H              ; 1'S COMPLEMENT H,L
  6271 2F          330 CMA
  6272 67          331 MOV H,A
  6273 7D          332 MOV A,L
  6274 2F          333 CMA
  6275 6F          334 MOV L,A
  6276 1600        335 MVI D,0              ; D,E = 1
  6278 1E01        336 MVI E,1
  627A 19          337 DAD D                ; ADD D,E TO H,L
  627B D1          338 POP D                ; POP OFF STACK
  627C C9          339 RET                  ; END SUBROUTINE
                   340
                   341 END

PUBLIC SYMBOLS


EXTERNAL SYMBOLS


USER SYMBOLS
ADM    A 6203   BINOM  A 6060   CNTR   A 6086   CNTR2  A 60BD   CNTR3  A 6172   CNTR4  A 61B3   COMP   A 623D
COMP2  A 6245   COMP3  A 624B   CONT   A 6089   CONT2  A 60C0   CONT3  A 6175   CONT4  A 61B6   DEN    A 605B
DENM1  A 605F   DENPN  A 605D   FINAL  A 61FA   INCR   A 60A9   INCR2  A 60E0   INCR3  A 619A   INCR4  A 61DB
INSRT  A 622F   MLTY   A 61F5   MOD1   A 626F   MPY    A 6253   MPY2   A 6263   MPYL   A 6027   MPYM   A 6028
MPYS   A 6029   MULT   A 61EA   NUM    A 605A   NUMM1  A 605E   NUMPN  A 605C   ONE    A 61FE   RPT    A 60FA
RPT2   A 6133   SET1   A 6215   SET2   A 6221   SHFT   A 6104   SHFT2  A 613D   TABLE  A 6000   TMP1   A 6024
TMP2   A 6026   X2NA   A 60EE   X2NB   A 612E   XFORM  A 6080   XFORM2 A 60B7   XFORM3 A 6167   XFORM4 A 61A8

ASSEMBLY COMPLETE,  NO ERRORS
```

```
ISIS-II ASSEMBLER SYMBOL CROSS REFERENCE, V2.0                  PAGE   1

ADM      36    66   180   216   255#
BINOM    17#   26    56   170   206
CNTR     25#   51
CNTR2    55#   81
CNTR3   169#  198
CNTR4   205#  234
COMP    267   296#
COMP2   272   301#
COMP3   281   303#
CONT     26#   39
CONT2    56#   69
CONT3   170#  183
CONT4   206#  219
DEN       9#   67    79   181   196   217   232
DENM1    13#  160
DENPN    11#  202   223
FINAL   248#  253
INCR     44#   48
INCR2    74#   78
INCR3   191#  195
INCR4   227#  231
INSRT   283#  305
MLTY    245#  247
MOD1    297   301   303   328#
MPY      35    65   179   215   307#
MPY2    319#  321
MPYL      5#  268   322
MPYM      6#
MPYS      7#  269   316
MULT     28    58   172   208   238#
NUM       8#   37    49    84
NUMM1    12#  121
NUMPN    10#  166   187
ONE     243   252#
RPT      88#  123
RPT2    127#  162
SET1    268#  299
SET2    273#  302
SHFT     95#  109
SHFT2   134#  148
TABLE    16#   24    42    54    72    87   126   165   186   201   222
TMP1      3#   25    55   169   205   259   284
TMP2      4#   94   103   105   110   133   142   144   149
X2NA     83#
X2NB     86   125#
XFORM     2    23#
XFORM2   53#
XFORM3  164#
XFORM4  200#

CROSS REFERENCE COMPLETE
```

```
ASM80 DIFFEQ.SRC SYMBOLS XREF


ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        DIFFEQ   PAGE   1
DIFF. EQN. OUTPUT ROUTINE

 LOC  OBJ         SEQ        SOURCE STATEMENT

                    1 $TITLE ('DIFF. EQN. OUTPUT ROUTINE')
                    2 NAME DIFFEQ
6024                3 TMP1 EQU 6024H
6076                4 TMP3 EQU 6076H
6078                5 ADL EQU 6078H
6029                6 MPYS EQU 6029H
605C                7 NUMPN EQU 605CH
6045                8 YCOEFF EQU 6045H
6030                9 XCOEFF EQU 6030H
607C               10 ZXH EQU 607CH
607E               11 ZXS EQU 607EH
F80F               12 OUT1 EQU 0F80FH
6203               13 ADM EQU 6203H
6027               14 MPYL EQU 6027H
6000               15 TABLE EQU 6000H
                   16
                   17
6300               18 ORG 6300H
6300 217C60        19 DIFF: LXI H,ZXH       ; H,L GETS 607CH
6303 222460        20 SHLD TMP1             ; STORE H,L IN TMP1
6306 210000        21 LXI H,00              ; H,L SET TO ZERO
6309 227C60        22 SHLD ZXH
630C 3E00          23 MVI A,0H              ; A SET TO ZERO
630E 327E60        24 STA ZXS               ; 607EH SET TO ZERO
6311 310080        25 LXI SP,8000H          ; SET STACK POINTER
6314 213060        26 LXI H,XCOEFF          ; H,L GETS 6030H
6317 3A5C60        27 LDA NUMPN             ; SET H,L TO POINT AT OLDEST TIME
631A 85            28 ADD L                 ; FACTOR
631B 6F            29 MOV L,A
631C 227860        30 SHLD ADL              ; STORE H,L IN ADL
631F 210060        31 LXI H,TABLE           ; H,L GETS 6000H
6322 3A5C60        32 LDA NUMPN             ; SET H,L TO POINT AT CORRESPONDING
6325 85            33 ADD L                 ; COEFFICIENT
6326 6F            34 MOV L,A
6327 227660        35 SHLD TMP3             ; STORE H,L IN TMP3
632A 2A7660        36 BACK: LHLD TMP3       ; LOAD H,L FROM TMP3
632D 7D            37 MOV A,L
632E FEFD          38 CPI 0FDH              ; CHECK IF ALL COEFFICIENTS USED
6330 CA8363        39 JZ STG2
6333 56            40 MOV D,M               ; LOAD D,E WITH COEFFICIENT
6334 2C            41 INR L
6335 5E            42 MOV E,M
6336 2C            43 INR L
6337 7E            44 MOV A,M               ; A GETS SIGN BIT
6338 322960        45 STA MPYS
633B 2D            46 DCR L                 ; L GETS L - 5
633C 2D            47 DCR L
633D 2D            48 DCR L
633E 2D            49 DCR L
633F 2D            50 DCR L
6340 227660        51 SHLD TMP3             ; STORE H,L IN TMP3
6343 2A7860        52 LHLD ADL              ; LOAD H,L FROM ADL
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        DIFFEQ   PAGE   2
DIFF. EQN. OUTPUT ROUTINE

 LOC  OBJ         SEQ       SOURCE STATEMENT

 6346 7D           53 MOV A,L               ; CHECK FOR PRESENT TIME FACTOR
 6347 FE30         54 CPI 30H
 6349 CA7563       55 JZ INPX               ; INPUT X FACTOR IF TIME IS RIGHT
 634C 46           56 MOV B,M               ; LOAD B,C WITH INPUT FACTOR
 634D 2C           57 INR L
 634E 4E           58 MOV C,M
 634F 2C           59 INR L
 6350 7E           60 MOV A,M
 6351 227860       61 RERTE: SHLD ADL       ; STORE H,L IN ADL
 6354 C5           62 PUSH B
 6355 47           63 MOV B,A               ; ADJUST PRODUCT SIGN BIT
 6356 3A2960       64 LDA MPYS
 6359 A8           65 XRA B
 635A 322960       66 STA MPYS
 635D C1           67 POP B
 635E CDD764       68 CALL MPYDB            ; MULTIPLY COEFFICIENT BY X FACTOR
 6361 222760       69 SHLD MPYL
 6364 CD0362       70 CALL ADM              ; ADD MPY # TO SUMMATION
 6367 2A7860       71 LHLD ADL              ; LOAD H,L FROM ADL
 636A 2D           72 DCR L                 ; L GETS L - 5
 636B 2D           73 DCR L
 636C 2D           74 DCR L
 636D 2D           75 DCR L
 636E 2D           76 DCR L
 636F 227860       77 SHLD ADL              ; STORE H,L IN ADL
 6372 C32A63       78 JMP BACK              ; RETURN TO NEW INPUT
                   79
 6375 0680         80 INPX: MVI B,80H       ; SET B,C = 8000H
 6377 0E00         81 MVI C,0
 6379 3E00         82 MVI A,0               ; SET A = 0
 637B 70           83 MOV M,B               ; STORE A,B,C IN X FACTOR TABLE
 637C 2C           84 INR L
 637D 71           85 MOV M,C
 637E 2C           86 INR L
 637F 77           87 MOV M,A
 6380 C35163       88 JMP RERTE             ; RETURN TO INPUT ROUTINE
                   89
 6383 214560       90 STG2:LXI H,YCOEFF     ; H,L GETS 6045H
 6386 3A5C60       91 LDA NUMPN             ; SET H,L TO POINT AT OLDEST TIME
 6389 85           92 ADD L                 ; FACTOR
 638A 6F           93 MOV L,A
 638B 227860       94 SHLD ADL              ; STORE H,L IN ADL
 638E 211260       95 LXI H,TABLE+12H       ; H,L GETS 8012H
 6391 3A5C60       96 LDA NUMPN             ; SET H,L TO POINT AT CORRESPONDING
 6394 85           97 ADD L                 ; COEFFICIENT
 6395 6F           98 MOV L,A
 6396 227660       99 SHLD TMP3             ; STORE H,L IN TMP3
 6399 2A7660      100 BACK2: LHLD TMP3      ; LOAD H,L FROM TMP3
 639C 7D          101 MOV A,L
 639D FE12        102 CPI 12H               ; CHECK IF ALL COEFFICIENTS USED
 639F CAF663      103 JZ STG3
 63A2 56          104 MOV D,M               ; LOAD D,E WITH COEFFICIENT
 63A3 2C          105 INR L
 63A4 5E          106 MOV E,M
 63A5 2C          107 INR L
```

```
 LOC  OBJ         SEQ        SOURCE STATEMENT

 63A6 7E          108 MOV A,M               ; A GETS SIGN BIT
 63A7 322960      109 STA MPYS
 63AA 2D          110 DCR L                 ; L GETS L - 5
 63AB 2D          111 DCR L
 63AC 2D          112 DCR L
 63AD 2D          113 DCR L
 63AE 2D          114 DCR L
 63AF 227660      115 SHLD TMP3             ; STORE H,L IN TMP3
 63B2 2A7860      116 LHLD ADL              ; LOAD H,L FROM ADL
 63B5 46          117 MOV B,M               ; B,C GETS OUTPUT FACTOR
 63B6 2C          118 INR L
 63B7 4E          119 MOV C,M
 63B8 2C          120 INR L
 63B9 7E          121 MOV A,M
 63BA 227860      122 SHLD ADL              ; STORE H,L IN ADL
 63BD C5          123 PUSH B
 63BE 47          124 MOV B,A               ; ADJUST SIGN OF PRODUCT
 63BF 3A2960      125 LDA MPYS
 63C2 A8          126 XRA B
 63C3 2F          127 CMA
 63C4 E680        128 ANI 80H
 63C6 322960      129 STA MPYS
 63C9 C1          130 POP B
 63CA CD0764      131 CALL MPYDB            ; MULTIPLY COEFFICIENT BY Y FACTOR
 63CD 222760      132 SHLD MPYL
 63D0 C3E463      133 JMP CHK               ; CHECK FOR ZERO PRODUCT
 63D3 CD0362      134 CONT: CALL ADM        ; ADD TO SUMMATION
 63D6 2A7860      135 RRTE2: LHLD ADL
 63D9 2D          136 DCR L                 ; L GETS L - 5
 63DA 2D          137 DCR L
 63DB 2D          138 DCR L
 63DC 2D          139 DCR L
 63DD 2D          140 DCR L
 63DE 227860      141 SHLD ADL              ; STORE H,L IN ADL
 63E1 C39963      142 JMP BACK2             ; GO FOR NEW COEFFICIENT
                  143
 63E4 7D          144 CHK: MOV A,L          ; CHECK H,L FOR ZERO NUMBER
 63E5 FE00        145 CPI 0
 63E7 C2F363      146 JNZ PASS
 63EA 7C          147 MOV A,H
 63EB FE00        148 CPI 0
 63ED C2F363      149 JNZ PASS
 63F0 C3D663      150 JMP RRTE2
 63F3 C3D363      151 PASS: JMP CONT
                  152
                  153
 63F6 2A7660      154 STG3: LHLD TMP3       ; LOAD H,L FROM TMP3
 63F9 46          155 MOV B,M               ; B,C GETS Y OUTPUT COEFFICIENT
 63FA 2C          156 INR L
 63FB 4E          157 MOV C,M
 63FC 2C          158 INR L
 63FD 7E          159 MOV A,M
 63FE C5          160 PUSH B
 63FF 217C60      161 LXI H,ZXH             ; LOAD H,L WITH SUMMATION # LOCATION
 6402 56          162 MOV D,M               ; LOAD D,E WITH SUMMATION NUMBER
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        DIFFEQ   PAGE    4
DIFF. EQN. OUTPUT ROUTINE

 LOC  OBJ          SEQ        SOURCE STATEMENT

 6403 2C            163 INR L
 6404 5E            164 MOV E,M
 6405 2C            165 INR L
 6406 46            166 MOV B,M                  ; ADJUST SIGN BIT
 6407 A8            167 XRA B
 6408 324760        168 STA YCOEFF+2             ; STORE SIGN BIT
 640B C1            169 POP B
 640C 310288        170 LXI SP,8802H             ; SET STACK POINTER
 640F CDF664        171 CALL DIVDB               ; DIVIDE D,E BY B,C
 6412 214560        172 LXI H,YCOEFF             ; H,L GETS 6045H
 6415 70            173 MOV M,B                  ; STORE QUOTIENT IN MEMORY
 6416 2C            174 INR L
 6417 71            175 MOV M,C
 6418 C5            176 PUSH B
 6419 78            177 MOV A,B                  ; OUTPUT QUOTIENT TO OUTPUT DEVICE
 641A CDB264        178 CALL HEX1
 641D C1            179 POP B
 641E 79            180 MOV A,C
 641F CDB264        181 CALL HEX1
 6422 3A4760        182 LDA YCOEFF+2
 6425 CDB264        183 CALL HEX1
 6428 0E0D          184 MVI C,0DH                ; OUTPUT CARRIAGE RETURN
 642A CD0FF8        185 CALL OUT1
 642D 0E0A          186 MVI C,0AH                ; OUTPUT LINE FEED
 642F CD0FF8        187 CALL OUT1
                    188
                    189
 6432 014560        190 MOVE: LXI B,YCOEFF       ; H,L GETS 6045H
 6435 3A5C60        191 LDA NUMPN                ; SET H,L TO POINT AT OLDEST TIME FACTOR
 6438 C602          192 ADI 02H
 643A 81            193 ADD C
 643B 4F            194 MOV C,A
 643C 3C            195 INR A
 643D 3C            196 INR A
 643E 3C            197 INR A
 643F 5F            198 MOV E,A
 6440 50            199 MOV D,B
 6441 0A            200 SHFTDN: LDAX B           ; MOVE ALL TIME FACTORS DOWN TO
 6442 12            201 STAX D                   ; NEXT TIME FACTOR SLOT
 6443 0D            202 DCR C
 6444 1D            203 DCR E
 6445 7B            204 MOV A,E
 6446 FE2F          205 CPI 02FH
 6448 CA4E64        206 JZ TIMER                 ; JUMP TO TIMER WHEN FINISHED
 644B C34164        207 JMP SHFTDN
                    208
                    209
                    210
 644E 210080        211 TIMER: LXI H,8000H
 6451 2B            212 TMOUT: DCX H
 6452 7D            213 MOV A,L
 6453 FE00          214 CPI 0
 6455 C25164        215 JNZ TMOUT
 6458 7C            216 MOV A,H
 6459 FE00          217 CPI 0
```

```
 LOC  OBJ        SEQ       SOURCE STATEMENT

 645B C25164      218 JNZ TMOUT
 645E C30063      219 JMP DIFF
 0050             220 DS 50H
                  221
                  222 ;END OF PROGRAM
                  223
 64B1 76          224 HLT
                  225 ; OUTPUT HEX NUMBERS
 64B2 F5          226 HEX1: PUSH PSW          ; PUSH REGISTER ON STACK
 64B3 1F          227 RAR                     ; RIGHT SHIFT 4 TIMES
 64B4 1F          228 RAR
 64B5 1F          229 RAR
 64B6 1F          230 RAR
 64B7 E60F        231 ANI 0FH                 ; A GETS A LOGICAL AND OFH
 64B9 C630        232 ADI 30H                 ; A GETS A + 30H
 64BB FE3A        233 CPI 3AH
 64BD FAC264      234 JM OUT2                 ; CHECK IF A LESS THAN 3AH
 64C0 C607        235 ADI 07H                 ; ADD 7 IF NOT
 64C2 4F          236 OUT2: MOV C,A
 64C3 CD0FF8      237 CALL OUT1               ; OUTPUT MOST SIGNIFICANT PART
 64C6 F1          238 POP PSW                 ; POP REGISTER OFF STACK
 64C7 E60F        239 ANI 0FH                 ; A GETS A LOGICAL AND OHF
 64C9 C630        240 ADI 30H                 ; A GETS A + 30H
 64CB FE3A        241 CPI 3AH                 ; CHECK IF A LESS THAN 3AH
 64CD FAD264      242 JM OUT3
 64D0 C607        243 ADI 07H                 ; ADD 7 IF NOT
 64D2 4F          244 OUT3: MOV C,A
 64D3 CD0FF8      245 CALL OUT1               ; OUTPUT LEAST SIGNIFICANT PART
 64D6 C9          246 RET                     ; END OF SUBROUTINE
                  247
                  248
                  249 ; MULTIPLY/DIVIDE SUBROUTINE
                  250
 64D7 210000      251 MPYDB: LXI H,00         ; H,L SET TO ZERO
 64DA 3E10        252 MVI A,16                ; A = 16
 64DC F5          253 MPY2: PUSH PSW
 64DD 7B          254 MOV A,E                 ; CHECK IF LSB IS ZERO
 64DE E601        255 ANI 01H
 64E0 CAE464      256 JZ MPY1
 64E3 09          257 DAD B                   ; ADD B,C TO H,L IF LSB ≠ ZERO
 64E4 7C          258 MPY1: MOV A,H           ; RIGHT SHIFT H,L AND D,E
 64E5 1F          259 RAR
 64E6 67          260 MOV H,A
 64E7 7D          261 MOV A,L
 64E8 1F          262 RAR
 64E9 6F          263 MOV L,A
 64EA 7A          264 MOV A,D
 64EB 1F          265 RAR
 64EC 57          266 MOV D,A
 64ED 7B          267 MOV A,E
 64EE 1F          268 RAR
 64EF 5F          269 MOV E,A
 64F0 F1          270 POP PSW
 64F1 3D          271 DCR A                   ; A GETS A - 1
 64F2 C2DC64      272 JNZ MPY2                ; CONTINUE MULTIPLY IF A ≠ 0
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0          DIFFEQ  PAGE   6
DIFF. EQN. OUTPUT ROUTINE

 LOC  OBJ         SEQ        SOURCE STATEMENT

 64F5 C9           273 RET                      ; END SUBROUTINE
                   274
 64F6 37           275 DIVD0: STC               ; CARRY SET
 64F7 3F           276 CMC                      ; CARRY CLEARED
 64F8 7A           277 MOV A,D                  ; RIGHT SHIFT D,E
 64F9 1F           278 RAR
 64FA 57           279 MOV D,A
 64FB 7B           280 MOV A,E
 64FC 1F           281 RAR
 64FD 5F           282 MOV E,A
 64FE 37           283 DIVD1: STC               ; SET CARRY
 64FF 3F           284 CMC                      ; CLEAR CARRY
 6500 78           285 MOV A,B                  ; RIGHT SHIFT AND 1'S COMPLIMENT B,C
 6501 1F           286 RAR
 6502 2F           287 CMA
 6503 47           288 MOV B,A
 6504 79           289 MOV A,C
 6505 1F           290 RAR
 6506 2F           291 CMA
 6507 4F           292 MOV C,A
 6508 03           293 INX B                    ; B GETS B + 1
 6509 210000       294 LXI H,00                 ; H,L SET TO ZERO
 650C 3E11         295 MVI A,17                 ; A = 16
 650E EB           296 XCHG                     ; SWAP D,E WITH H,L
 650F E5           297 DV0: PUSH H
 6510 09           298 DAD B                    ; ADD B,C TO H,L
 6511 DA1565       299 JC DV1                   ; CHECK FOR CARRY BIT
 6514 E1           300 POP H                    ; RESTORE OLD H,L IF NO CARRY
 6515 F5           301 DV1: PUSH PSW
 6516 7B           302 MOV A,E                  ; LEFT SHIFT D,E AND H,L
 6517 17           303 RAL
 6518 5F           304 MOV E,A
 6519 7A           305 MOV A,D
 651A 17           306 RAL
 651B 57           307 MOV D,A
 651C 7D           308 MOV A,L
 651D 17           309 RAL
 651E 6F           310 MOV L,A
 651F 7C           311 MOV A,H
 6520 17           312 RAL
 6521 67           313 MOV H,A
 6522 F1           314 POP PSW
 6523 3D           315 DCR A                    ; A GETS A - 1
 6524 C20F65       316 JNZ DV0                  ; CONTINUE DIVISION IF A ≠ 0
 6527 7A           317 MOV A,D                  ; PLACE D,E IN B,C
 6528 47           318 MOV B,A
 6529 7B           319 MOV A,E
 652A 4F           320 MOV C,A
 652B 310088       321 LXI SP,8800H             ; SET STACK POINTER
 652E C9           322 RET                      ; END SUBROUTINE
                   323
                   324
                   325
                   326 END
```

```
USER SYMBOLS


ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        DIFFEQ   PAGE    7
DIFF. EQN. OUTPUT ROUTINE

ADL    A 6078   ADM    A 6203   BACK   A 632A   BACK2  A 6399   CHK    A 63E4   CONT   A 63D3   DIFF   A 6300
DIVD1  A 64FE   DIVDB  A 64F6   DV0    A 650F   DV1    A 6515   HEX1   A 64B2   INPX   A 6375   MOVE   A 6432
MPY1   A 64E4   MPY2   A 64DC   MPYDB  A 64D7   MPYL   A 6027   MPYS   A 6029   NUMPN  A 605C   OUT1   A F80F
OUT2   A 64C2   OUT3   A 64D2   PASS   A 63F3   RERTE  A 6351   RRTE2  A 63D6   SHFTDN A 6441   STG2   A 6383
STG3   A 63F6   TABLE  A 6000   TIMER  A 644E   TMOUT  A 6451   TMP1   A 6024   TMP3   A 6076   XCOEFF A 6030
YCOEFF A 6045   ZXH    A 607C   ZXS    A 607E

ASSEMBLY COMPLETE,  NO ERRORS
```

REFERENCES

1. Close, C.M.; DeRusso, P.M.; and Roy, R.J. State Variables For Engineers. New York: John Wiley and Sons, 1965.

2. Cohen, D., and Simons, F.O., Jr. "An In-Place Algorithm for Computing the Bilinear Transform of Polynomials." Unpublished research paper, California: University of Southern California / Information Sciences Institute, 1978.

3. Harden, R.C., and Simons, F.O., Jr. "Differential Equation Solutions For Up to 10th Order System Theory Models With H.P. - 67 Compulators." Unpublished research paper, Florida: University of Central Florida, 1978.

4. INTEL 8080 Assembly Language Programming Manual. California: INTEL Corp., 1976.

5. Stanley, W.D. Digital Signal Processing. Virginia: Reston Publishing Company, Inc., 1975.

6. Wavell, R.B. "Microcomputers: An Alternative for Digital Controllers." Unpublished Masters thesis, University of Central Florida, 1979.