NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

REAL TIME PROGRAMMING
OF A ROBOT

by

Paulo R. Souza

December 1986

Thesis Advisor G. J. Thaler

CURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
DECLASSIFICATION / DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION aval Postgraduate School	6b OFFICE SYMBOL (If applicable) 32
7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
ADDRESS (City, State, and ZIP Code) onterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93940-5000
NAME OF FUNDING / SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

TITLE (Include Security Classification)
REAL TIME PROGRAMMING OF A ROBOT

PERSONAL AUTHOR(S)
Souza, Paulo R.

TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
aster's Thesis	FROM ______ TO ______	1986 December	118

SUPPLEMENTARY NOTATION

COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Robot - Real Time Programming Autoadaptive model

ABSTRACT (Continue on reverse if necessary and identify by block number)

Two difficulties that arise in controlling a robot arm (plant) are the changes in inertia and the lack of a velocity feedback. The inertia of the arm varies when the robot picks up or releases a load and the velocity would need a tachometer to be measured (expensive and not practical). One way to overcome those problems is to use an autoadaptive model to represent the plant. If the model "follows" the plant transfer function and both have the same input, the model can have velocity feedback and the effects will be reflected in the plant. The solution presented above was investigated and simulated in DSL by Kenneth R. Wikstrom, in his thesis from NPS in September of 1986. In the present research, a hardware and assembly software was designed and implemented based on the same structure mentioned in that thesis. The block diagram and autoadaptive algorithm were slightly modified and the plant was simulated in a dedicated analog computer. Two transfer functions were tested in the analog plant: a disk drive motor and a robot motor.

DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
NAME OF RESPONSIBLE INDIVIDUAL G. J. Thaler	22b TELEPHONE (Include Area Code) 408-6247980	22c OFFICE SYMBOL 62Tr

FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete
SECURITY CLASSIFICATION OF THIS PAGE

Real Time Programming of a Robot

by

Paulo R. Souza
Captain, Brazilian Air Force
B.S., Instituto Tecnologico de Aeronautica, Brazil, 1979

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1986

ABSTRACT

Two difficulties that arise in controlling a robot arm (plant) are the changes in inertia and the lack of a velocity feedback. The inertia of the arm varies when the robot picks up or releases a load and the velocity would need a tachometer to be measured (expensive and not practical). One way to overcome those problems is to use an autoadaptive model to represent the plant. If the model "follows" the plant transfer function and both have the same input, the model can have velocity feedback and the effects will be reflected in the plant. The solution presented above was investigated and simulated in DSL by Kenneth R. Wikstrom, in his thesis from NPS in September of 1986. In the present research, a hardware and assembly software was designed and implemented based on the same structure mentioned in that thesis. The block diagram and autoadaptive algorithm were slightly modified and the plant was simulated in a dedicated analog computer. Two transfer functions were tested in the analog plant: a disk drive motor and a robot motor.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The positioning of a robot arm with just one degree of freedom can be a hard task if some constraints are imposed by the problem. Some of the requirements and/or restrictions can be:

1. Accuracy.
2. Minimum time.
3. No overshoot.
4. Velocity measurements are not available.
5. Loads can be changed during operation.

The accuracy and overshoot are common requirements and will not be discussed. The minimum time requirement is,in a way, incompatible with the lack of a tachometer to provide velocity measurements. How can the motor achieve the required position with no overshoot and minimum time without the knowledge of the velocity along the trajectory? It cannot be done unless the velocity can be guessed using the position information that is readily available. In this case the curve following method can be used and the arm will be decelerated after some suitable point in the trajectory.

What is the effect of changes in the loads? They will modify the inertia of the system and the transfer function of the plant. So, a real time identification algorithm is needed to detect inertia changes and update the controller.

With these constraints and solutions in mind let us analyze the scheme proposed in Figure 1.1 . The plant represents the arm and motor of the robot. The model is a very simple approximation of the plant. Both model and plant are driven by the same input. The identification algorithm receives both outputs, identifies the actual plant parameters and updates the model in order to keep CM and CS as close as possible.

The theoretical scheme was treated in [Ref. 1] and will not be detailed here. However, the block diagram will be explained in Chapter II. The purpose of this research was to implement the real time system using a microprocessor, in a protoboard level. The model and controller were implemented by software in the microprocessor and the plant was simulated using analog hardware.

When the desired position is achieved the controller, that was in "bang-bang" mode, can be switched to a linear compensator. This feature is important but was not

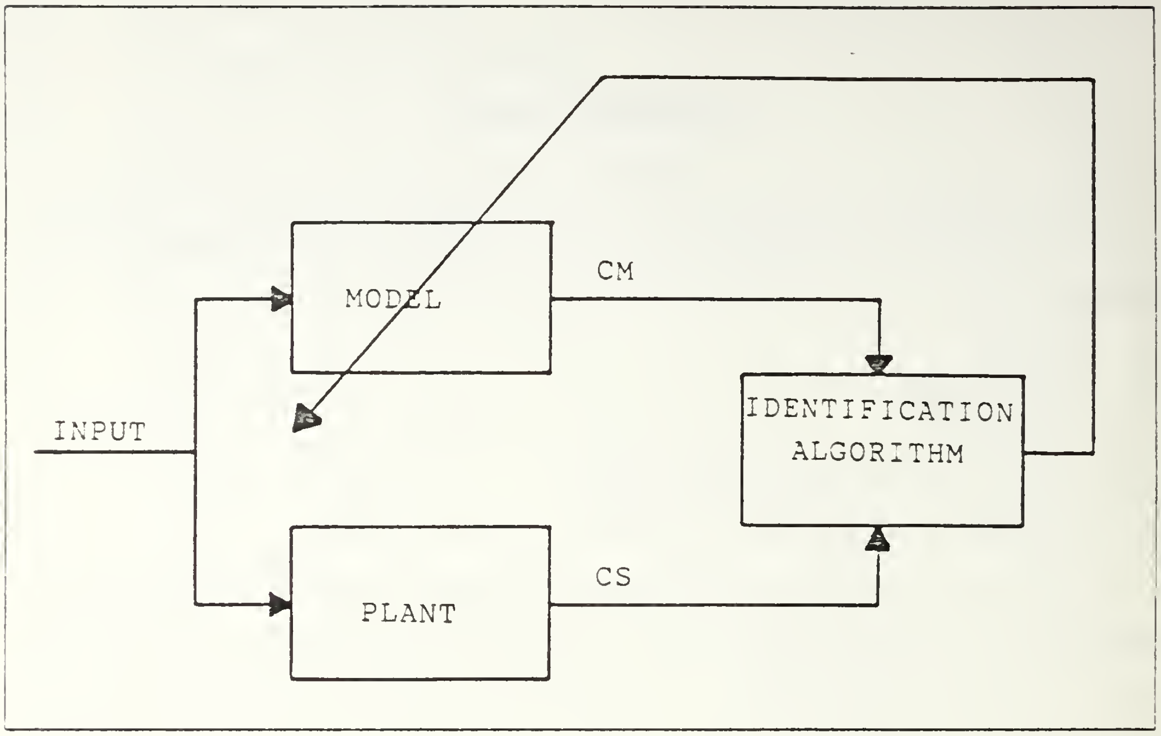

Figure 1.1 Simplified block diagram of the system.

implemented in this thesis since our goal was to prove the feasibility of the autoadaptive algorithm using a servo system as a controller.

The identification algorithm was chosen to be efficient and simple. The algorithm used in this thesis is found in [Ref. 1] and will be referred as "The Wikstrom Algorithm". It will be briefly discussed in Chapter II and the software implementation will be presented in Chapter III (subroutine "Walg").

The proposed system studied on a block diagram basis is presented in Chapter II. The 16 bits software design is discussed in Chapter III. Chapter IV presents the hardware for the analog plant and the microprocessor. The results are presented in Chapter V. The conclusions and some possible areas to be studied are treated in Chapter VI. The 16 bit program (called "Model") and the monitor program are presented in Appendices A and B, respectively.The procedures to operate the hardware are discussed in Appendix C.

The Z-80 microprocessor was chosen to implement the system for three main reasons:

1. The monitor was already developed and, with small changes, could provide all the support needed to run and debug the software.

2. The author is very familiar with this microprocessor.

3. The time requirements to operate a robot arm are not a constraint to the use of this microprocessor. On the other hand, the idea behind the model.controller and algorithm is still valid for a faster microprocessor.

All the software was developed using the Z-100 and Data I/O in the Digital Lab. The programs were stored on floppy disks, under CP/M operational system, and transferred to EPROM's via Data I/O. Once in EPROM's they were installed in the protoboards and then executed using the monitor. The debugging process was done using some features of the monitor that will be explained later on.

The microprocessor and interfaces treated in this research can be considered as individual parts of a large system. Suppose a robot with multiple arms, mechanically coupled by joints. Each arm will have a terminal microprocessor (TM). This microprocessor implements the model, controller and updating algorithm accomplished in this thesis. The position input for each arm is commanded by a central computer. This command is applied directly to the corresponding Terminal Microprocessor, as shown in Figure 1.2 .

The inputs to the central computer (CC) are the actual and desired (future) position of each arm. Based on this data it calculates the best trajectory for each arm and sends the individual position command for the TM's. The central computer does not worry about changes in the inertias or other factors that can occur in the individual arms. These problems are solved by the terminal microprocessors.

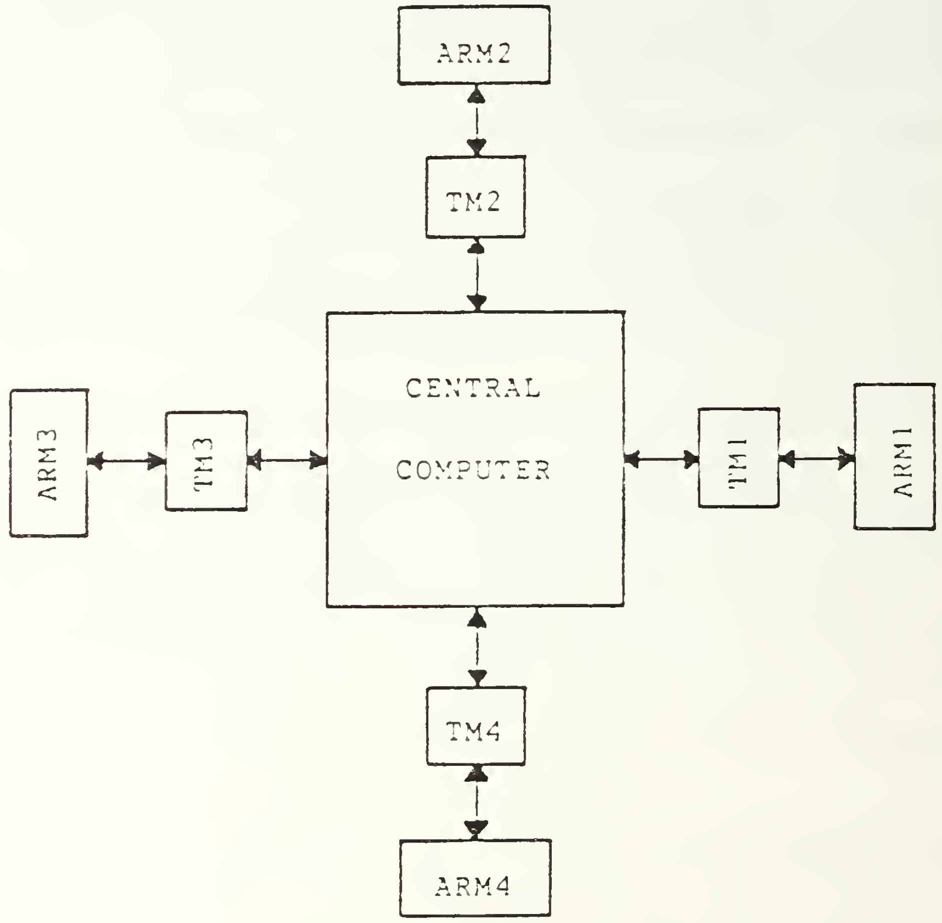

Figure 1.2 The Central Computer and the Terminal Microprocessors.

II. THE PROPOSED SYSTEM

A. INTRODUCTION

The entire system can be separated in three different areas: model, plant and algorithm, as presented in Figure 1.1 . However, physically, there are two main parts: the digital microprocessor that contains the model and algorithm, and the analog implementation of the plant that represents the robot arm.

The whole system is depicted in Figure 2.1 and the only difference between this one and that presented in [Ref. 1] is the input of the plant. In [Ref. 1] the plant input is XDOTE and here is V. This choice simplifies the hardware of the plant and still keeps the requirement that both model and plant have to have identical inputs.

B. MODEL AND CONTROLLER

The simple model that tries to approximate the plant consists of two integrators and a constant gain, Km, as shown in Figure 2.2 . The servo mechanism with velocity curve following and "bang bang" amplifier that was shown in Figure 2.1 belongs to the controller.

Now that the controller is separated from the model, let us go back to Figure 2.1 and examine every block. The input of the regulator, R, is a step command that determines the angular movement of the robot arm. The error, E, is the input of the curve . The curve approximates the deceleration curve of the motor and is a parabola. Mathematically, we have

$XDOT = A\sqrt{E}$,where $A = K1\sqrt{2.Km.Vsat}$

Vsat= saturation limit of the amplifier

Thus, from the servo error and the knowledge of the system, the desired velocity , XDOT, is computed by taking the square root of the error and multiplying by A.This velocity is compared with the actual velocity, CDOT, to generate the velocity error, XDOTE, that drives the amplifier (limiter). If the actual voltage, CDOT, is less than the theoretical one, XDOT, the limiter is set to +Vsat (+10 volts), that means, the system keeps applying "full power" to get to the desired position. The curve following process is illustrated in Figure 2.3 . When CDOT reaches XDOT the deceleration process starts and the limiter alternates -Vsat and +Vsat at its output in a bang bang process. The gain K2 is very large to guarantee the full acceleration.

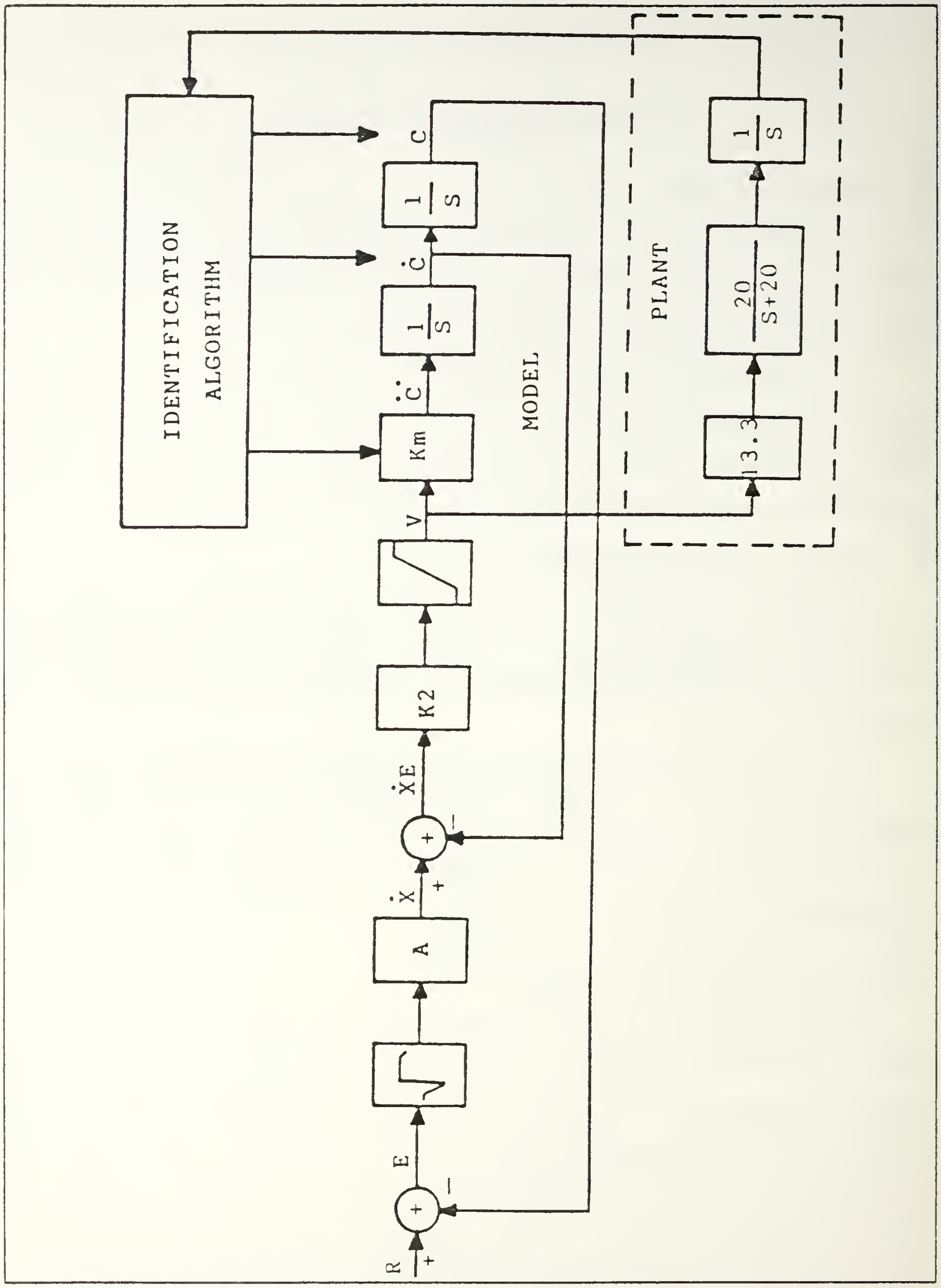

Figure 2.1 The Proposed System.

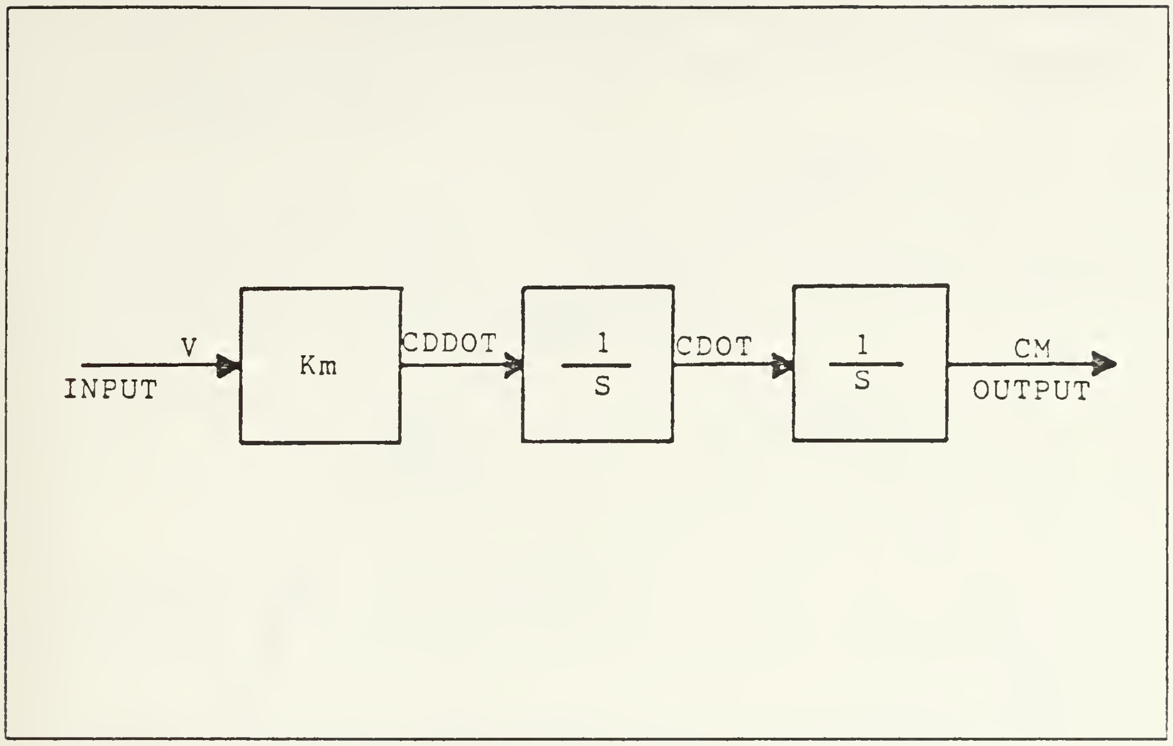

Figure 2.2 The simple model.

C. THE PLANT

In this research two plants were tested, a disk driver and a robot motor. The transfer function of the disk driver was implemented first and in order to compare with the results obtained in [Ref. 1] the motor and load parameters were assumed to be the same, that is, the motor of a disk driver and a very small arm. As presented in Figure 2.1 it is a second order system with a mechanical pole at 20.55 rad/sec , a pole at the origin and a gain equal to 13.3.

D. THE IDENTIFICATION ALGORITHM

The identification algorithm is represented in a separate block in Figure 2.1 because it has a specific function and "connects" plant and model. However, it is a piece of software included in the microprocessor and imbedded in the program Model.

The specific function of the algorithm is to measure the plant output and update the model parameters in such a way that the model approximates the plant the best it can. Since both model and plant have the same input, if they have identical transfer functions they will have the same behavior. This is what we need because the model is

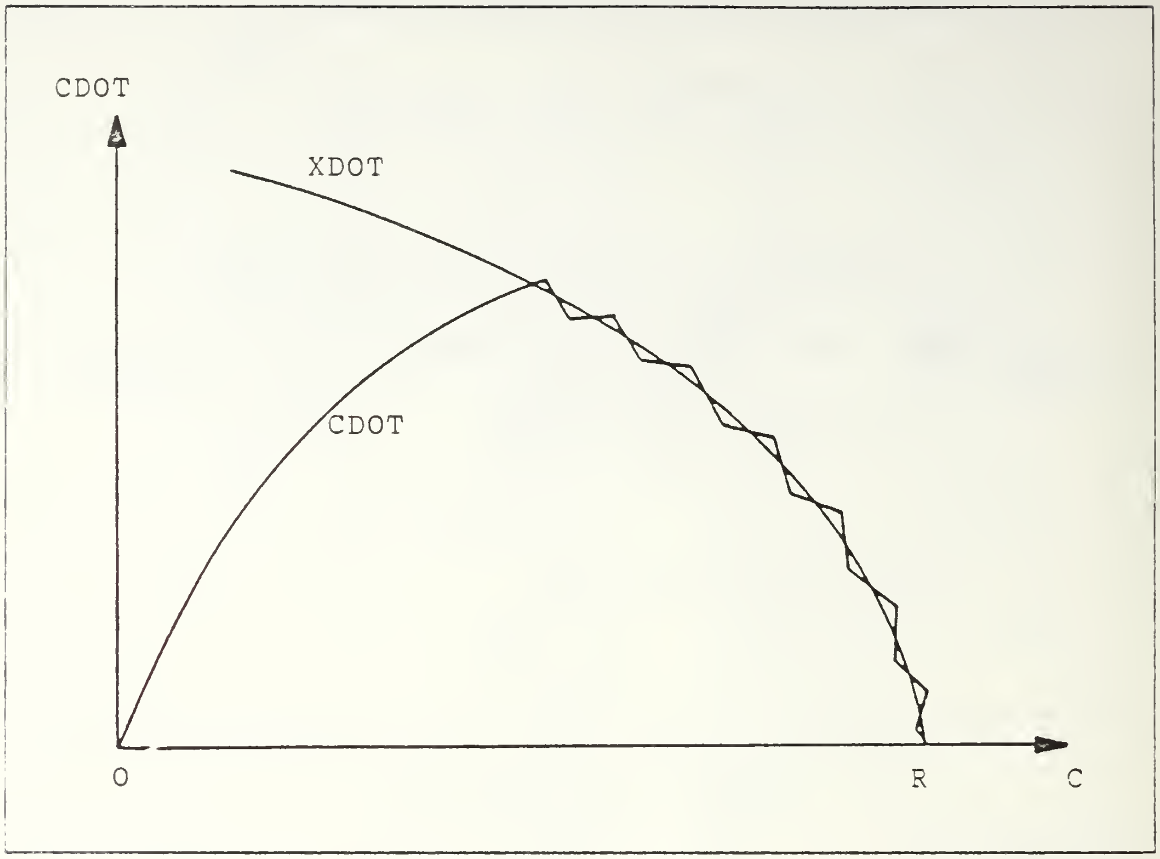

Figure 2.3 Curve Following Process.

perfectly controllable with position and velocity feedback. Thus, controlling the model, the plant will be controlled.

In an ideal case the model will be exactly the same as the plant during the entire trajectory. This was the case when, as an experience to observe the behavior of the analog plant, it was set up as a double integrator and a gain, becoming a copy of the model. The results were fine and will be presented in detail in Chapter IV, where the plant hardware is discussed.

As mentioned in Chapter I the algorithm used in this thesis was developed by Wikstrom in [Ref. 1] for a disk driver system whose transfer function is that of the plant studied in Chapter IV. Wikstrom considered two mandatory requirements to implement the autoadaptive algorithm:

1. The calculations must be reasonably accurate to allow the model states to approximate the trajectory of the servo motor during the seek mode. The seek mode occurs in the full acceleration phase of the trajectory.

2. The calculations must be simple to minimize the computation time.

In the algorithm, the gain Km, the model output CS and the model velocity CDOT are updated. In the ideal model if C is the position and Vsat the saturation voltage of the amplifier we can write:

$$C = Km.V_{sat}(t^2/2) \rightarrow Km = 2C/V_{sat}.t^2$$

For a sampling interval equal to T, t = NT, where N is the number of sampling intervals that occured up to time t. Letting C = CS :

$$Km = \frac{2CS}{V_{sat}(NT)^2}$$

The velocity of the plant can be computed as

$$CSDOT = \frac{CS - CS(N-1)}{T}$$

Or, more accurately

$$CSDOT = \frac{2[CS - CS(N-1)]}{T} - CSDOT(N-1)$$

III. SOFTWARE DESIGN - 16 BITS

A. INTRODUCTION

As pointed out in Chapter II, the model is just a double integrator with a constant gain Km. The controller is the servo system with position and velocity feedback and a velocity curve following. The block diagram of this system is depicted in Figure 2.1 .

In order to simplify the task for the microprocessor the amplifier was assumed to be just a limiter without any linear region and the gain K2 that appears in Figure 2.1 is not needed anymore. The system will have a "bang-bang" control, that is, V will be + 10 or -10 depending upon the value of XDOTE.

The software for this digital servo was designed following the natural sequence of the servo itself, from R to CM. In order to keep a uniform notation, equations, block diagrams, flowcharts and programs have the same name for the corresponding variables. The only difference is that in flowcharts and programs the dot above the letter, indicating derivative, is literally written. For instance, $\dot{C}$ or CDOT represent the derivative of C. Another point that must be clarified is that the output of the model is called CM in the assembly program, rather than C, to avoid confusion with the C register. Thus, the derivative of CM should be CMDOT but is called CDOT to simplify the notation. The flowcharts presented in this Chapter are directly related with the 16 bit program "Model", presented in Appendix A. The instructions used in the program are based in [Ref. 2].

B. MAIN PROGRAM

The two first blocks of Figure 3.1 initialize the parallel port and some variables . The parallel port organizes the traffic between the discrete and continuous world involved in this research. The function of each port will be explained in detail in Chapter IV where the hardware is presented. The variables used in the main program are discussed in the following paragraphs.

The step input, R, is the position to be achieved by the robot arm. Its value can be entered from the keyboard and is requested by the program. The maximum value allowed is 127, limited by the eight bit A D converter. CM is the output of the model and represents its actual position. CM1 is the position at time t minus one, that is, the

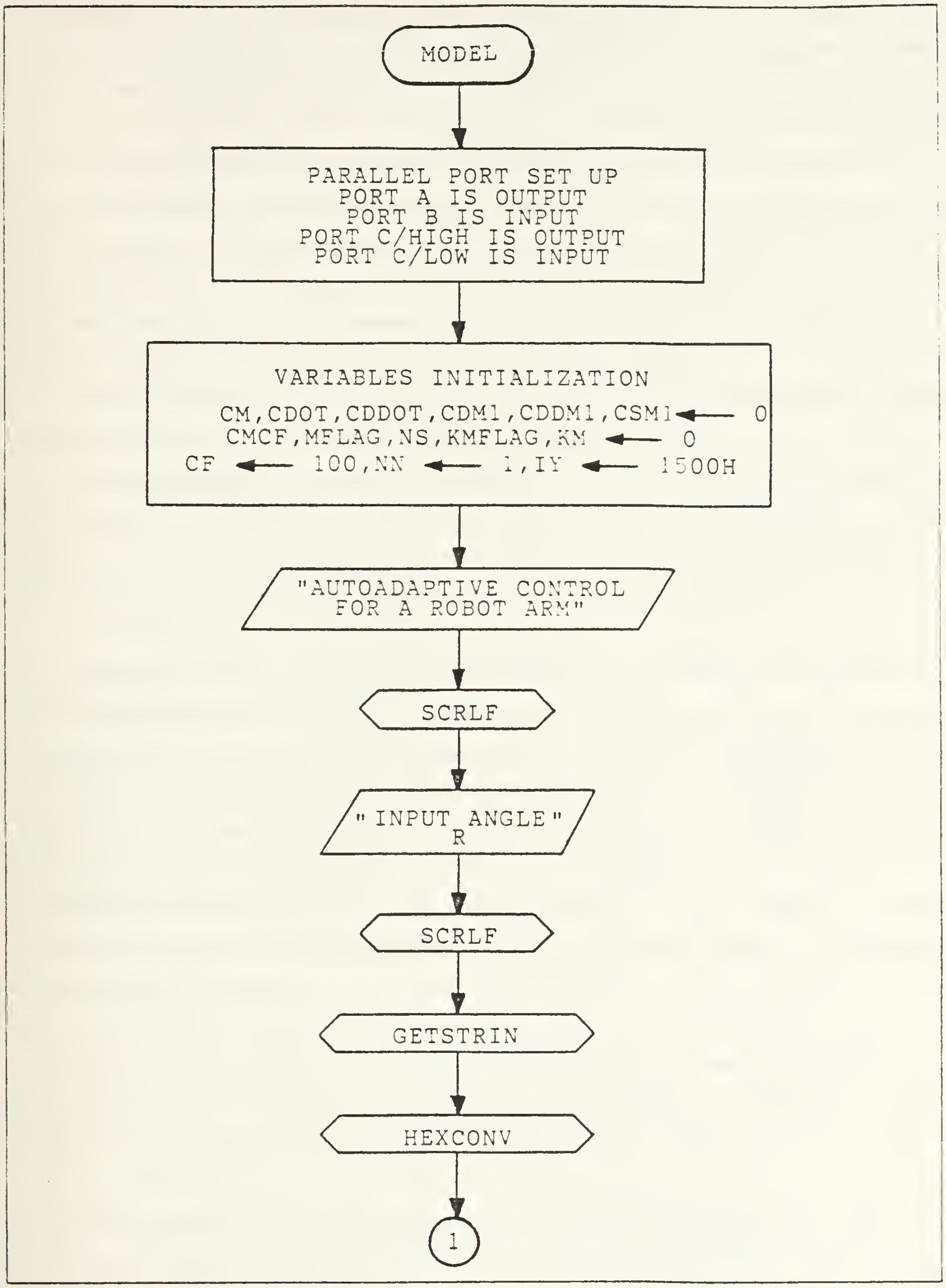

Figure 3.1 The Main Program - Flowchart A.

previous value of CM. The actual velocity of the model is CDOT. This velocity is fedback to be compared with XDOT as shown in Figure 2.1 . CDM1 is the model velocity at time t minus one or the last value of CDOT. CDDOT is the actual acceleration of the model. CDDM1 is the previous value of the acceleration.

The output of the plant is called CS and represents the actual position of the robot arm. CMCF represents the product between CM and CF, where CF is a correction factor used to raise the value of CM before integration. This will be explained later in this Chapter. XDOT is the velocity computed on the basis of the deceleration curve of the motor, as mentioned in Chapter II. XDOTE is the difference between the desired and actual velocity of the model. KM is the motor gain. N is a temporary memory for the index register IY. This register points to the next position of the memory available for data storage and its value is copied into the memory location N in order to check the end of memory. The memory available to store data goes from 1600H to 23FFH. So, the storage process must finish when the most significant byte of IY reaches 24H. NN counts the number of sampling intervals which have occured up to the present time and is used in the Wikstrom algorithm to compute the value of Km.

During the development of the software, all important variables were presented on the screen on-line, using a routine called WRITE. This routine was called after an operation as, for instance, an addition, and displayed the result stored in the register pair HL. In this fashion, the following variables were monitored in each loop: Error, square root of error, XDOT, XDOTE, CDDOT, CDOT, CM and CS. They were presented on the screen in hexadecimal codes. The only problem with this routine is that more than 400 microseconds were required to display one variable and this caused too much delay to the program .When the analog plant is part of the system we cannot afford such delays since the continuous system does not stop working. The alternative solution is to display the partial results off-line using the routine DISPLAY. But, before using this routine, the data must be stored in the memory. Thus, the routine WRITE was replaced by the routine STORE. This routine takes the value of the variable stored in the register pair HL and stores it into the data memory (from 1600H to 23FFH).

In some cases we do not need to store the variables in each consecutive loop but, for instance, store them each fifth loop. Thus, a flag must tell the STORE routine when a data is to be stored. This is the role of MFLAG. If MFLAG is zero the data is stored and when it is different than zero no data is stored. NS is the variable which controls the number of loops that will not be displayed. When the memory is full the flag is disabled and no more data is stored. However, the program continues running normally. The operation scheme of MFLAG is shown in Figure 3.2 . At the beginning of the program MFLAG is initialized to zero. So,during the first loop the data is stored by routine STORE. At the end of the loop NS is incremented and checked. Since its value is not 5 yet, MFLAG becomes FFH and no data is stored in the next four loops. In the fifth loop NS is 5 and the flag is inverted, becoming zero again and the sixth loop has data storage. But NS is set to zero again and the process is repeated until the memory available for data storage is full. At this point the flag is converted to 55H and from this point the inversion that occurs when NS is 5 does not convert the flag to zero anymore. That is, no more data is stored.

The main program is structured with many subroutines. Some of them belong to the monitor and will be briefly discussed here. The monitor, whose program is presented in Appendix B, is the operational system that supervises the microcomputer. The remaining subroutines will be discussed in the next paragraphs.

The subroutine STRING (monitor) receives characters from the keyboard and stores them in a stack . The subroutine HEXCONV (monitor) converts the data stored in the stack from ASCII to hexadecimal and stores them into the register pair DE. The subroutine SCRLF (monitor) provides carriage return and line feed.

The subroutines SUBTRACT, ADDITION, MULTIPLY, DIVIDE and CURVE when performed store the result in the register pair HL. The operands must be in register pairs DE and HL before the subroutine is called. In the case of the subroutine CURVE (input is the position error and output is XDOT), the operand must be in HL. In the subroutine SUBTRACTION the operand stored in DE is subtracted from the operand stored in HL. In the subroutine DIVIDE the operand stored in HL is divided by the operand stored in DE.

In order to be stored by subroutine STORE the data must be in register pair HL. This is very convenient since all operations send the result to these registers and all important variables come up from some operation.

All variables and subroutines that appear in Figure 3.1 were discussed in the previous paragraphs. Summarizing the operation of these first blocks :

1. The Parallel interface is set up
2. Variables are initialized

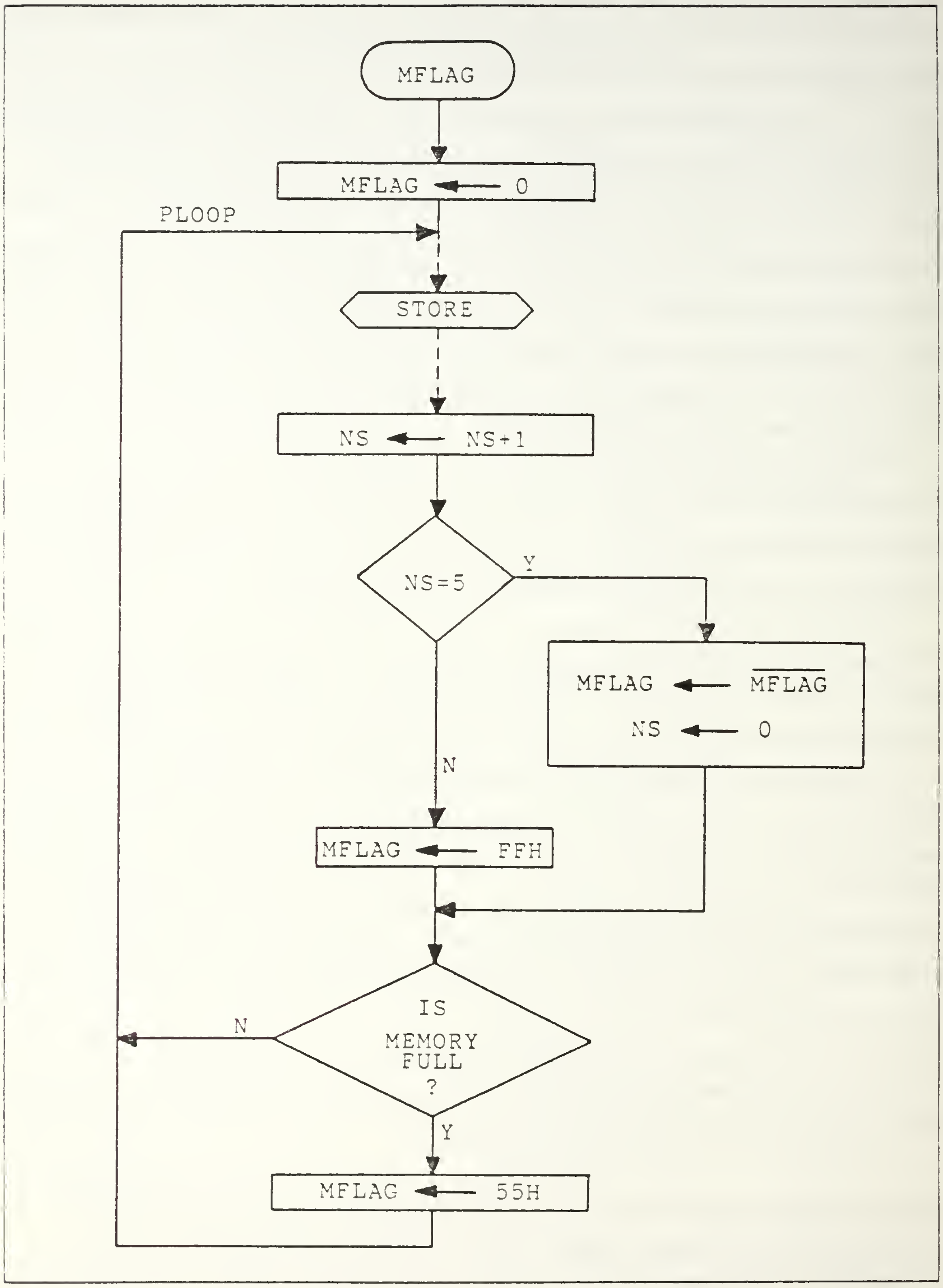

Figure 3.2 Data Storage Control.

3. Messages are sent to the screen introducing the system and asking for the angle input (R).
4. GETSTRIN gets the characters from the keyboard. HEXCONV transforms them from ASCII to hexadecimal.

The main program continues in the flowchart of Figure 3.3 . The value of the input position R is saved as an hexadecimal quantity. Between this block and the next is located the position feedback of the servo loop. The actual position is subtracted from the input position and the result is stored for future presentation (off-line). Register C is set as a flag to indicate if the position error is positive or negative. The error is tested. If it is negative, C is set to 1 and the number is converted to positive. If it is positive, no action is taken. XDOT is computed by subroutine CURVE and the result is stored into the register pair HL.

The flowchart of Figure 3.4 starts with register C being tested. If it is zero, the error was positive and no action is taken. If it is different than zero, the error was negative and XDOT is converted to negative. In both cases, XDOT is stored for future presentation on the screen. The actual velocity, CDOT is subtracted from XDOT resulting XDOTE, which is stored.

Continuing the description of the main program, the blocks of Figure 3.5 can be described in the following way. XDOTE, computed in the last operation is available in HL. Its value is checked. If it is positive, the variable V is set to +Vsat and if it is negative, V is set to -Vsat and the flag KMFLAG is set to 1. The value of V is the input of the model and also the input of the analog plant and must be sent to the digital to analog converter. The gain Km is multiplied by V and the result is stored.

The product of V and Km just computed represents the acceleration (CDDOT) of the model and its integration yields the velocity . Before discussing the flowchart that shows this computation, let us explain briefly the trapezoidal integration used in this program. Mathematically we have CDOT = CDOT + (CDDM1 + CDDOT) T/2 , where the previous value of CDOT is added to the trapezoidal area formed by the actual and previous value of CDDOT and the time lapsed between them, as illustrated by the shaded area in Figure 3.6 .

A very important point in the integration process is the integration step, T. It was determined by simply replacing the instructions "JR PLUS" and "JR VOLTS" by "NOP" instructions. Thus, the ±10 Volts are sent to the analog input and the time interval can be measured with an oscilloscope. It turned out to be 1.1 msec.

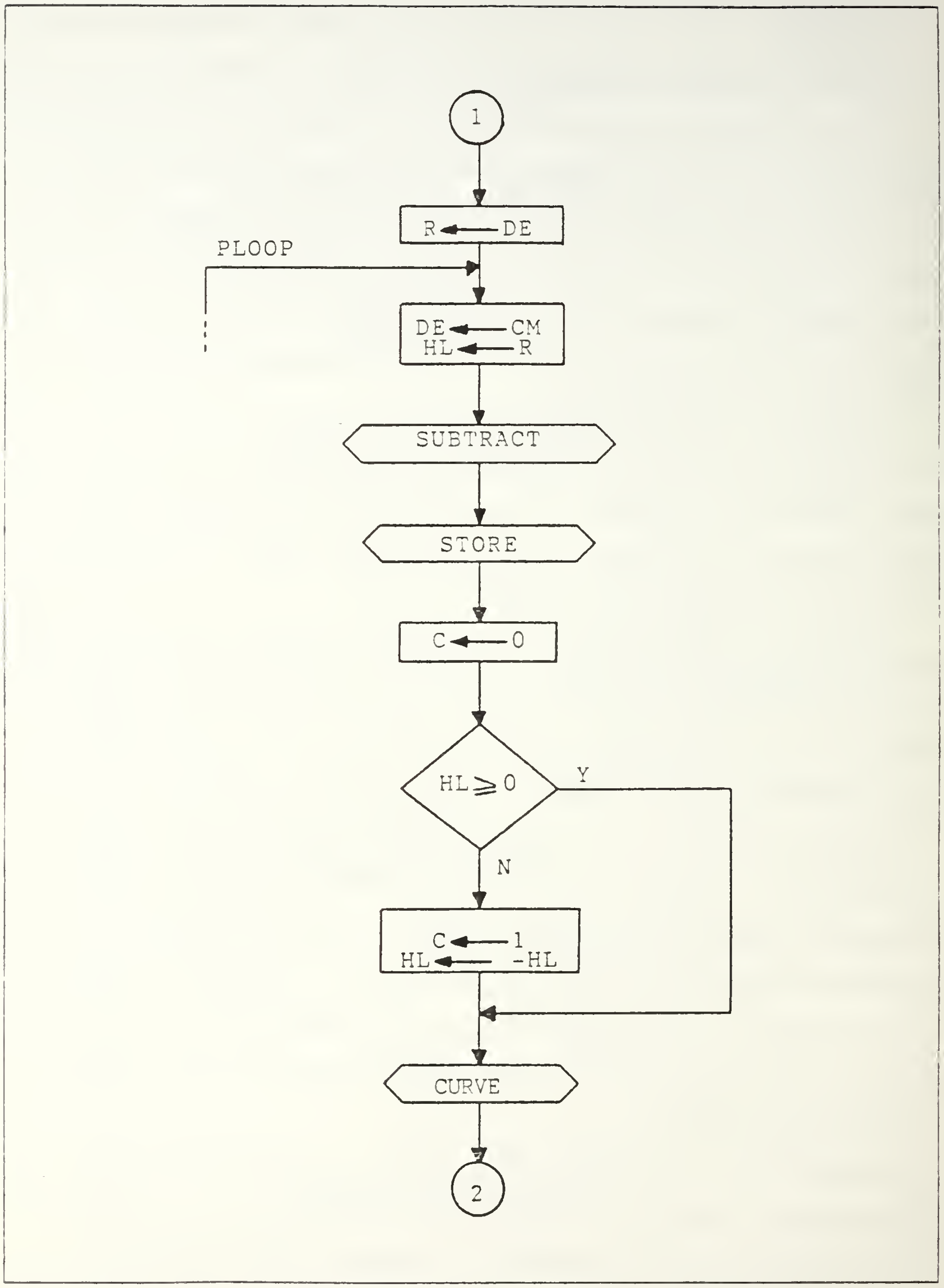

Figure 3.3 The Main Program - Flowchart B.

In Figure 3.7 the flowchart of the trapezoidal integration of the acceleration is presented and can be described as follows: The acceleration CDDOT , just computed, is saved. The previous value of CDDOT is loaded into the register pair DE and the actual value of CDDOT is stored in CDDM1, that is, CDDM1 is updated. The actual and previous value of the acceleration are added. The register pair DE receives T1 = 2/T. This transformation is necessary because T is a very small value and cannot be represented by a 16 bit integer number. So, instead of a multiplication by T/2, we can divide by 2/T = T1. The sum, CDDOT + CDDM1, is divided by T1. The last value of CDOT is added to the above result yielding: CDOT + (CDDOT + CDDM1) T/2 The actual value of CDOT is saved and stored for posterior presentation. This integration was first designed as a 16 bit computation and then modified to a 32 bit floating point program. The actual and previous value of the acceleration (CDDOT and CDDM1) are kept as 16 bit variables and the 32 bit value of CDOT is saved in temporary variables to be used in the next step. CDOT is also saved as a 16 bit number to be displayed on the screen.

The conversions from integer to floating point and vice-versa are not detailed in the flowchart but each number that goes to the Arithmetic Processing Unit is converted to a 32 bit floating point number before the operation.

The integration of CDOT just computed gives the actual value of the position, CM. However, there is a computational problem due to the fact that the integration step is very small. Since the first values of CDOT are also small and the computer is working with integer variables, all values of CM that are less than one are rounded to zero and not accumulated by the trapezoidal integration:

$$CM = CM + (CDOT + CDM1)\,T/2$$

As can be seen from the above equation, as far as the second term (CDOT+ CDM1)T/2 is zero (or rounded to zero) the value of CM will stay at zero. The way to bypass this problem was to magnify the value of the variable to be integrated (CDOT) multiplying it by an amplifier factor before the integration. After the integration the output is attenuated by the same factor. As shown in Figure 3.8 the corrector factor (CF) was set up to 100 and the output of the integrating block is CMCF that stands for CM times CF. This was the first approach to compute CM and could be modified to work with 32 bit floating point numbers as in the CDOT computation. However, CM is being updated every loop and the result of this computation is not taken into account. If this is not the case, the integration instructions can be easily modified by looking at the CDOT computation.

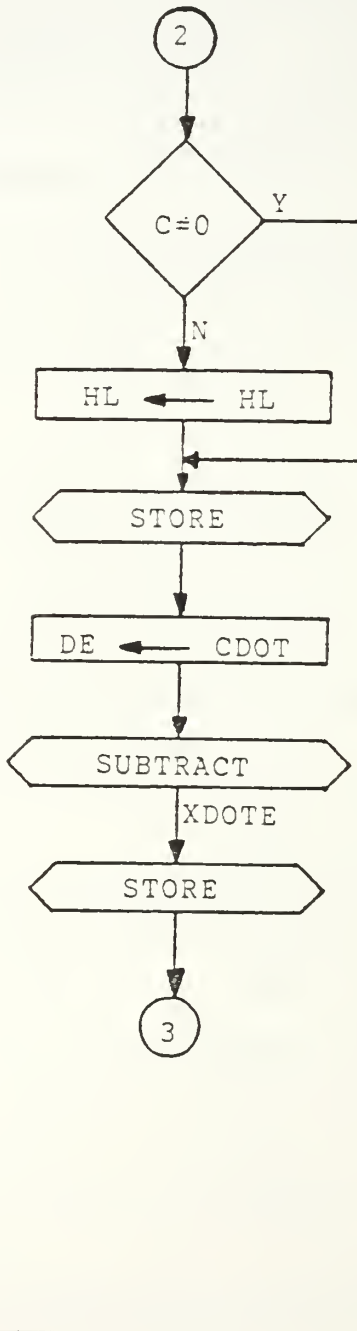

Figure 3.4 The Main Program Flowchart C.

With this introduction to the integration of the velocity the flowchart presented in Figure 3.9 is easily understood. This flowchart is a continuation of that studied in Figure 3.7 where CDOT was computed and loaded into the HL register. The last value of CDOT (CDM1) is loaded into the register pair DE and the actual value of CDOT updates CDM1 for the next loop. CDOT (in HL) and CDM1 (in DE) are added. The register pair DE is loaded with T1OCF that stands for T1 over CF. This parameter is created by the fact that, as mentioned in the first integration, T1 = 2/T and the corrector factor multiplies the input variables. The division rather than multiplication was explained in the last flowchart. The last value of CMCF is loaded into DE and added with (CDOT + CDM1)(T/2)CF yielding the new value of CMCF. CMCF is saved, DE is loaded with CF and after the division the new value of CM is obtained, that is CM = CMCF/CF.

The main program continues in the flowcharts presented in Figures 3.10 and 3.11 The idea behind this section of the program is to get the plant output and, using the Wikstrom algorithm, update the parameters of the model. Since these two flowcharts are at the end of the main program they also test the variables that control the end of the memory available for data storage and the frequency of storage . The flowcharts can be described as follows. CM, the actual model output just computed, is saved for future calculations and stored for off-line presentation. The subroutine ANALOG loads the plant output into the register pair HL and the subroutine STORE saves it at the data memory. The subroutine WALG applies the Wikstrom algorithm to update Km and CM based upon the actual and previous plant outputs.

The variable NS, whose function is to control the frequency of storage, is incremented and checked. In the example presented in the flowchart the desired frequency of storage is 5. This means that the data are to be storage every five loops. As explained before during the discussion of the MFLAG scheme, if NS = 5, MFLAG is complemented and becomes zero, enabling the storage process and NS is set to zero to restart the procedure. On the other hand, if NS is not zero MFLAG is set to FFH, disabling the storage process. The data memory is checked. If it is full MFLAG is set to 55H and from this point no more data is stored, since the complement of 55H is AAH and MFLAG will never be zero again. If the memory is not full, no action is taken. In both cases the program is addressed to the beginning to close the loop.

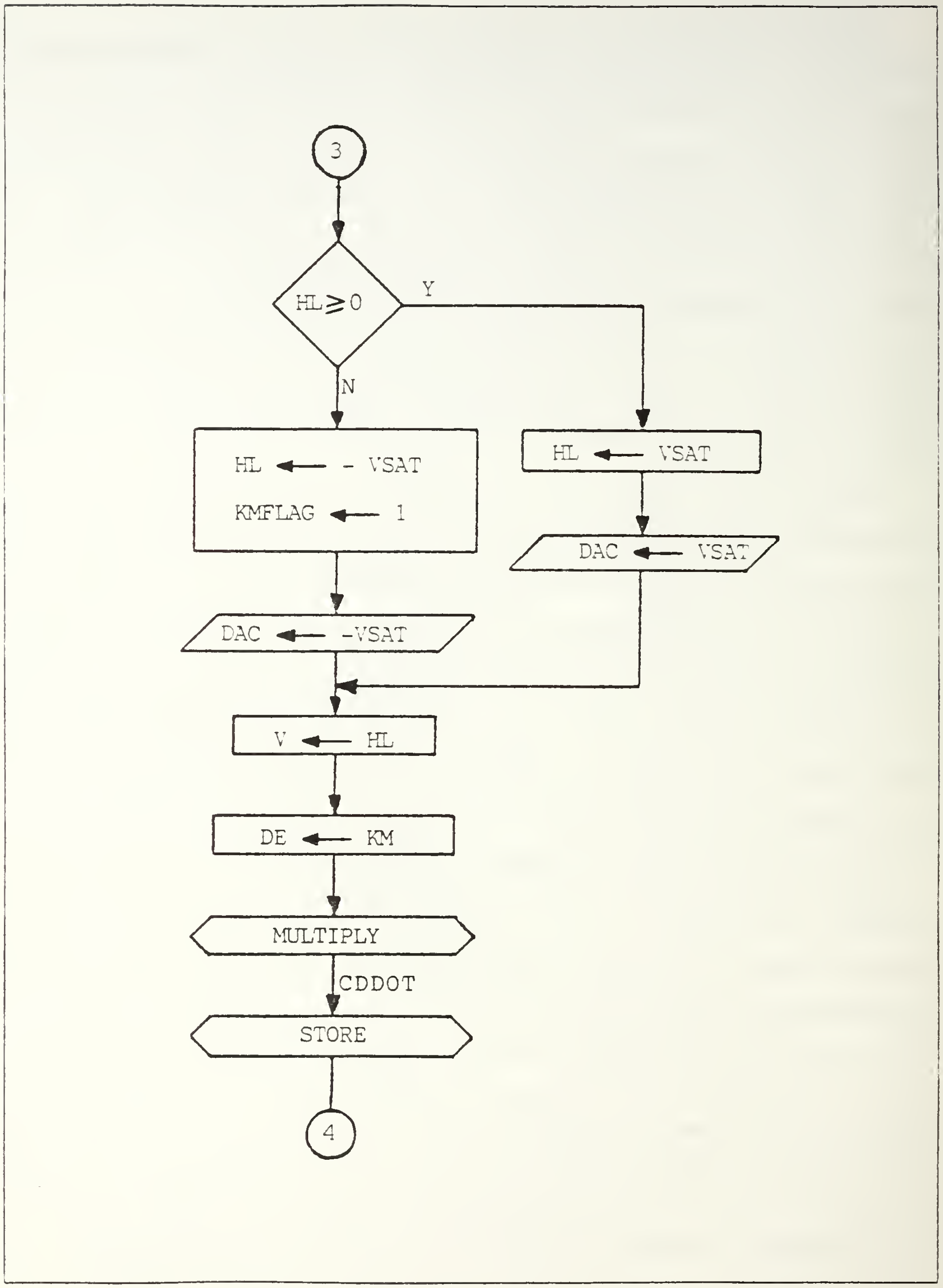

Figure 3.5 The Main Program -Flowchart D.

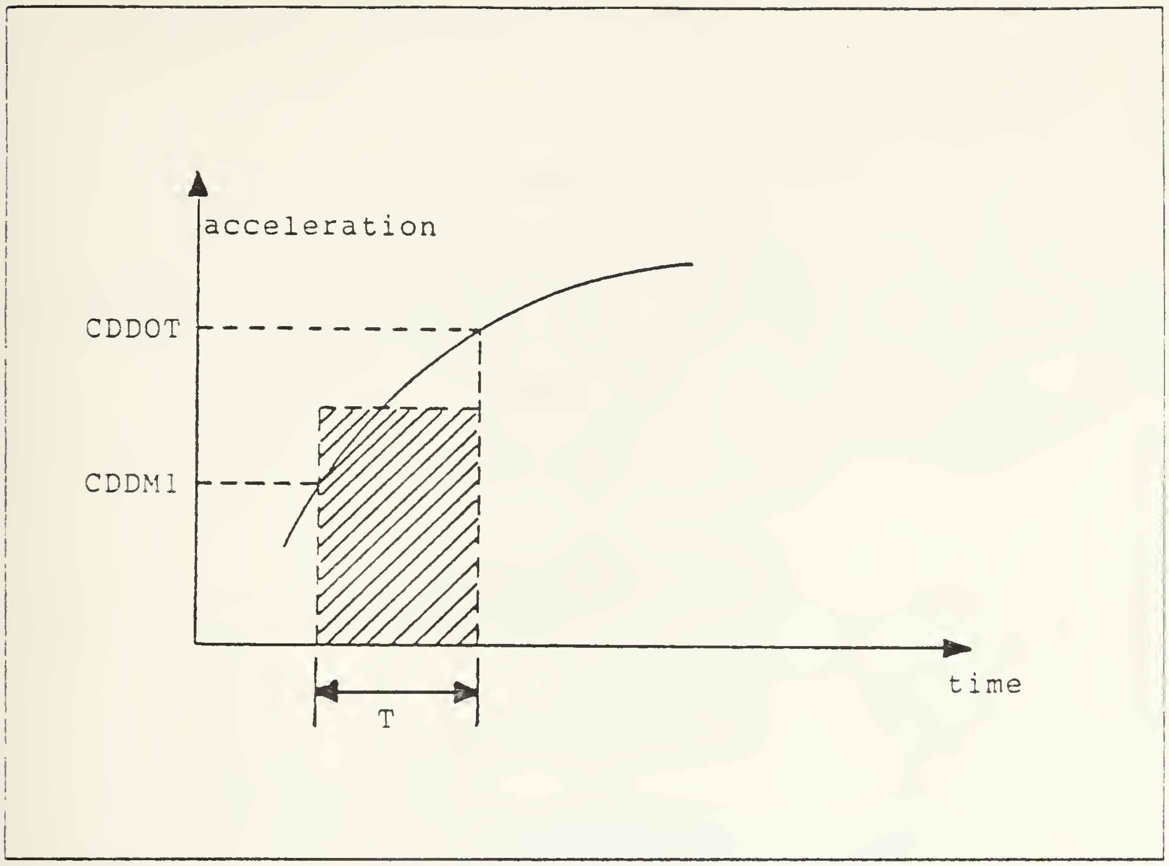

Figure 3.6 Trapezoidal Integration.

C. SUBROUTINES

The main program "MODEL" was structured to be easily understood and followed. Since it is written in Assembly language some simple operations need lots of instructions. In order to overcome this problem, all repetitive operations were implemented in subroutines.

Each time a subroutine is invoked, 13.5 microseconds are added to the program: 8.5 due to the instruction "CALL" and 5 microseconds due to the instruction "RET". In the present program there are 38 calls to subroutines yielding a total time of 513 microseconds. This reasonable "delay" is not affecting this experiment but could affect a practical application. However, if necessary, this time can be reduced to zero by just imbedding all the subroutines into the main program.

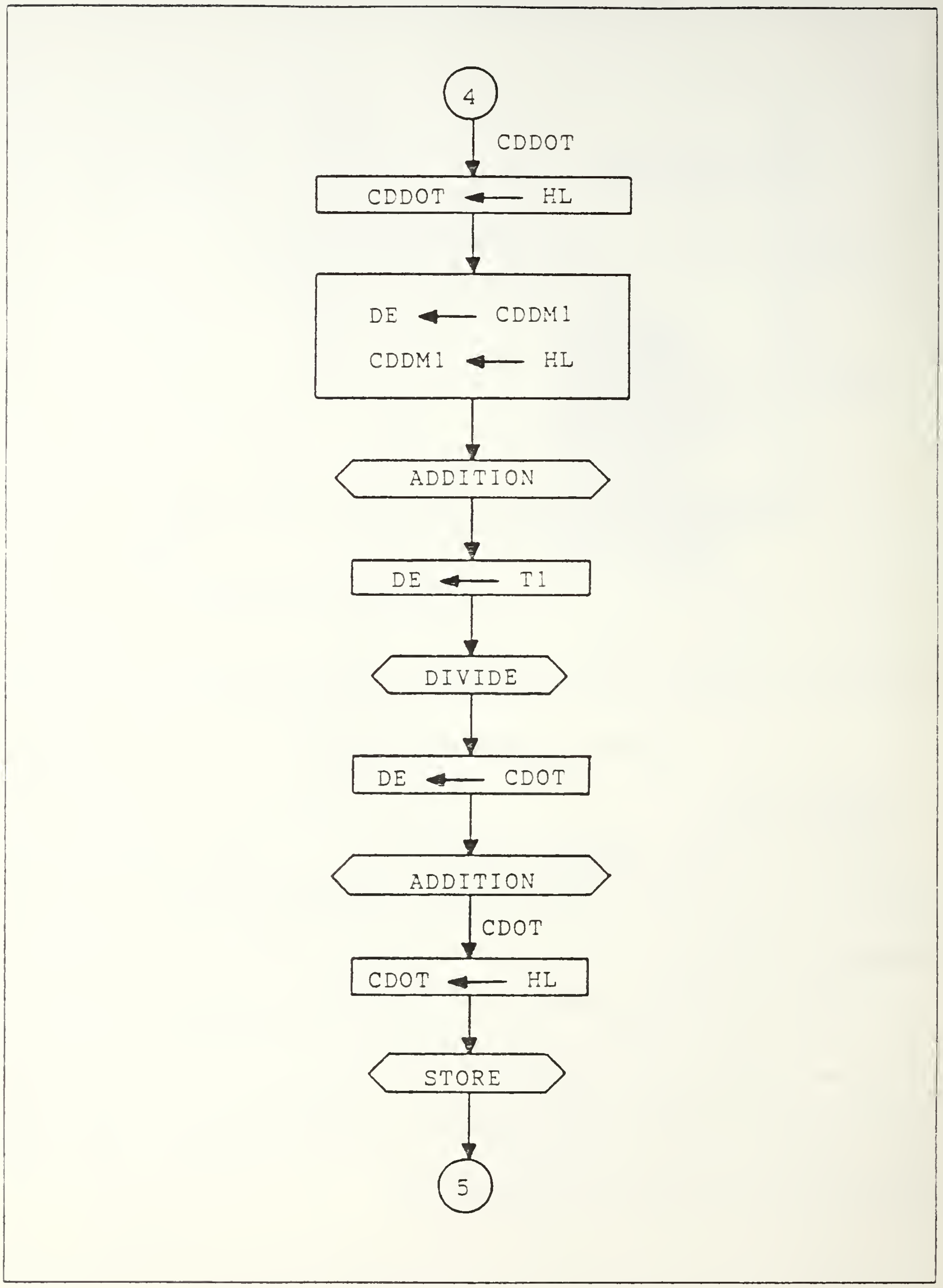

Figure 3.7 The Main Program - Flowchart E.

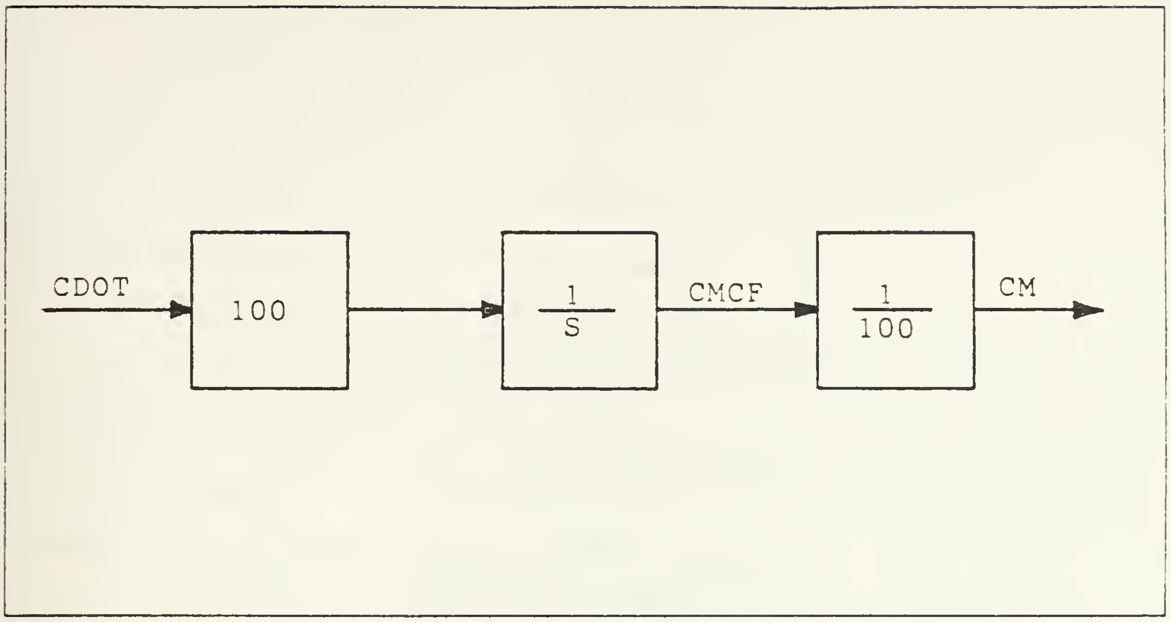

Figure 3.8 Velocity Integration using the Corrector Factor.

1. The Subroutine ANALOG

The objective of this subroutine is to sample the plant output, CS, at the end of each loop. This is done through port B of the parallel interface and the analog to digital converter (ADC). Port B of the interface is at address 02H as determined at the beginning of the main program.

Some of the instructions presented in this subroutine need a little knowledge of the hardware to be completely understood. However, the main idea can be absorbed from the flowchart presented in Figure 3.12 The blocks are now described in the sequence they appear in the flowchart. The ADC is enabled and starts converting the actual analog output to a corresponding eight bit number. The DATA READY pin of the ADC is polled and if data is available (conversion is completed), it is transferred to the A register. Otherwise the status is checked again. The incoming data is converted in a two's complement number by adding 80H. This is explained in detail in Chapter IV.

After conversion in a two's complement number the data sign is checked. If it is negative a conversion to positive is executed. The reason to justify this procedure is quite simple. In this research the movement of the robot arm is restricted to just one direction and negative numbers are not expected. However, at the beginning of the

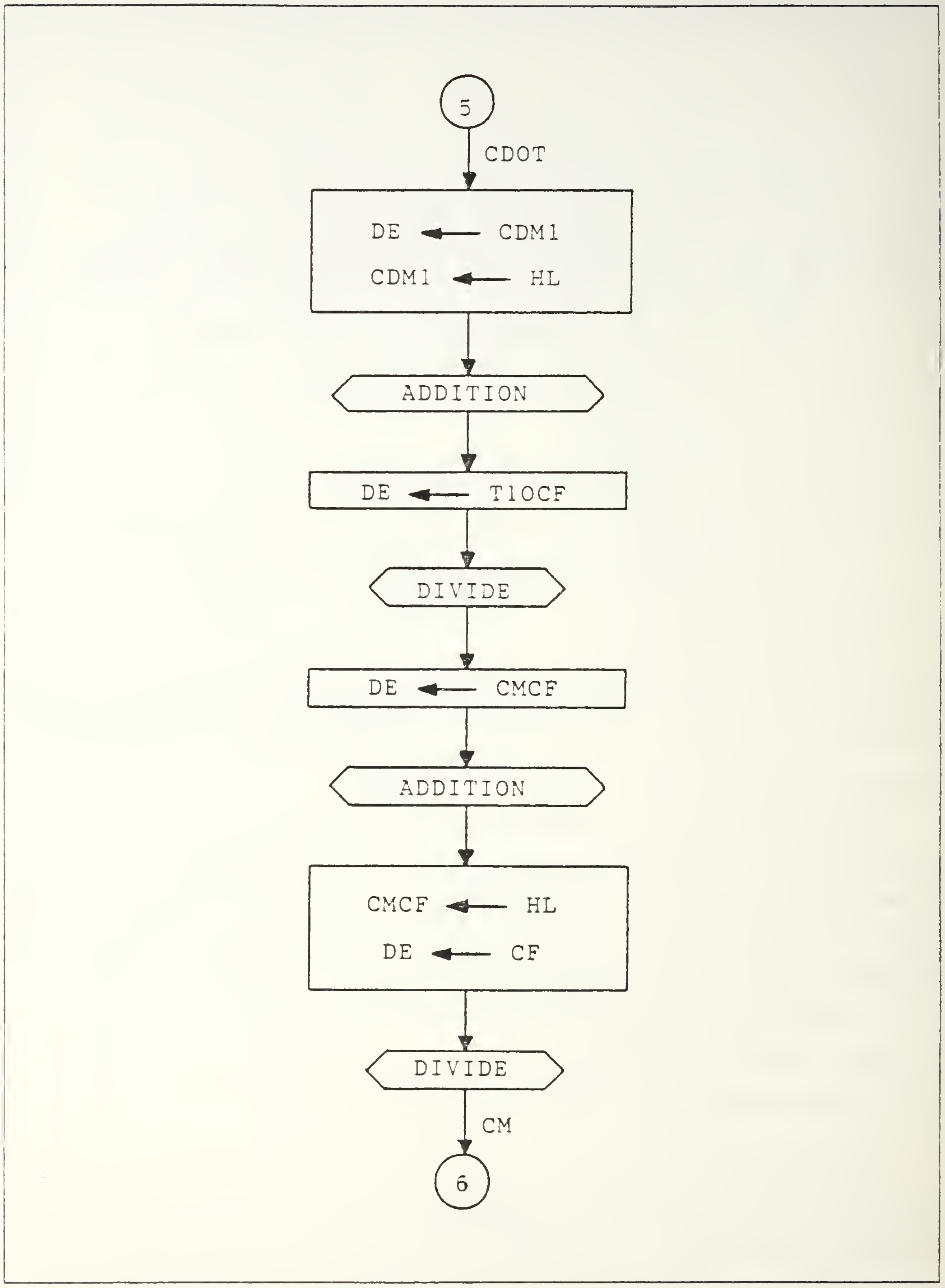

Figure 3.9 The Main Program - Flowchart F.

movement, when CS is still a very small number the noise can drive the output to a negative value and this would introduce an error in the system, since the program is not prepared to deal with negative numbers coming from the analog plant. Since the data is equal to or less than 127 and a positive number, it is loaded into register L and register H is set to zero. The ADC is disabled to allow the repetition of the process. This step will be detailed in Chapter IV.

2. The Subroutines of Basic Operations

The subroutines that perform addition, subtraction, multiplication and division will be called basic operations subroutines. They all use the Arithmetic Processing Unit (APU) Intel 8231 to perform the operations. The procedure is the same for these subroutines as shown in Figure 3.13 . The operands are sent to the APU by subroutine OUTOP. Then the appropriate command (each operation has its particular code) tells the APU what operation is to be performed with those operands. Finally, the subroutine INOP (input operand) retrieves the result from the APU. In all talks between APU and microprocessor the operands and commands must pass through register A of the CPU. The APU operands and commands are addressed to outputs 08H and 09H, respectively. An important point to be noticed in the program is that the operands must be loaded in register pairs HL and DE before calling a basic operation subroutine. In order to get correct results in the subtraction and division subroutines we have to keep in mind that the registers will be manipulated in the following way:

Subtraction is given by $HL \leftarrow HL - DE$

Division is given by $HL \leftarrow HL/DE$

3. The Subroutines OUTOP and INOP

As mentioned above the subroutines OUTOP and INOP communicate with the arithmetic processing unit in order to send operands before an operation and retrieve the result after the operation. The flowchart of the subroutine OUTOP is shown in Figure 3.14 . The contents of the registers HL and DE are sent to the APU in this order. Also notice that the low bytes are sent first in both cases. The subroutine INOP is presented in Figure 3.15 The result obtained in the APU after an operation is transfered to register A and then to registers H and L, in this order, one byte at a time. Notice that the high order byte of the sixteen bit result is the first to be retrieved.

4. **The Subroutine CURVE**

This subroutine uses the APU to compute XDOT from the position error and the constant SQRT($2.V_{sat}.Km$). The computation is all done inside the processing unit to avoid the error caused by converting the square root to an integer. Every variable or constant that is sent to the APU is converted to a 32 bit floating point number. After the computation XDOT is converted back to integer (refer to Figure 3.16).

5. **The Subroutine WALG**

This subroutine implements the autoadaptive algorithm described in [Ref. 1]. In that algorithm, the position, velocity and gain of the model are updated. However, the velocity CDOT is not being updated in this subroutine, for the following reason: the program was written for 16 bits but the interfaces were built to work with just eight bits, in order to simplify the plant. So, the biggest positive number the plant can deal with is 127 and the smallest is 1.

As a consequence of the above restrictions, the differences between an actual value of the plant output (CS) and its last value can not be always detected due to the round off problem. Since the computation of CDOT is based upon this difference, the result would be wrong most of the time. For instance, let's consider that the exact value of CS is 54.82 and the last value of the plant output (CSM1) was 54.17. Assuming that the velocity is computed using the simple formula CSDOT=(CS-CSM1)/T and working with integers the result will be CSDOT=0, since CS and CSM1 were both rounded to 54.

The flowchart of this subroutine is presented in Figure 3.17 and will be described in the following paragraphs. All the operations are done using the Arithmetic Processing Unit and integer numbers (16 bits).

The variable KMFLAG is set to zero at the beginning of the main program and when the full acceleration phase ends, that is CDOT is greater than XDOT for the first time, it is set to 1. In the subroutine, the flag is checked and if it is one, KM is not computed anymore, remaining with its last value.

In the full acceleration phase KM is computed and updated each time the subroutine is called. The formula used to compute KM is the following:

$$KM = 2CS\ Vsat(NT)^2 .$$

The values of Vsat and T are known and representing the constants by the parameter KMC we have:

$$KMC = 2/10Vsat(T)^2 = 16529 \text{ for } T = 1.1 \text{ ms and } Vsat = 10.$$

The extra constant in the denominator (10) was introduced because the numbers cannot be greater than 32767. So, after the division by N (represented by NN in the program) and the multiplication by CS, KMC must be multiplied by 10. This is shown in the eight blocks that follow the KMFLAG block ending with KM being updated. The reason KMC is not divided by NN^2 first (saving instructions) and then multiplied by CS and 10 is that when NN becomes a big number and KMC/NN^2 becomes less than one it is rounded to zero. So, after the first division (KMC/NN), there is a multiplication by CS to raise the result before dividing by NN again.

6. **The Subroutine STORE**

This subroutine transfers the data from register pair HL to the data memory. The purpose of this storage is to make those data available for an off-line presentation on the screen. As mentioned in the previous subroutines all operations send the result to the register pair HL. So, this routine is always called after some important operation. The flowchart is presented in Figure 3.18 and can be described as follows: The flag is checked and if it is zero the data is loaded into the memory and the pointer is incremented. Otherwise, no action is taken.

7. **The Subroutine DISPLAY**

The main purpose of this subroutine is the off-line presentation of the data stored by subroutine STORE on the screen off-line. That is, after the complete displacement of the robot arm, the program can be reset and the intermediate steps can be analyzed by calling the subroutine DISPLAY. The operation procedure is discussed in Appendix C. The variables that are set up to be stored by the actual program are: E, XDOT, XDOTE, CDDOT, CDOT, CM, CS and NN (the number of loops). They are stored from address 1600H to 23FFH, with a total of 3584 bytes. Since each variable has 16 bits and eight variables are stored each loop, the memory is able to store 224 loops of the main program. If the data are stored in consecutive loops, only the first 246 milliseconds of data will be stored. If all the loops are not stored this time can be increased. For instance, if the data are stored every fifth loop the program will be documented for 1.23 sec. The number of loops to be stored can be changed if the program is transferred to the RAM (refer to Appendix C).

Another important feature of this subroutine is the conversion from hexadecimal to decimal before displaying the numbers on the screen. This would be impossible to implement on line due to the amount of time required to implement this complex operation. The data is displayed in eight columns, ten rows at a time, and they can be scrolled up by pressing the space bar or any other key. The number of rows to be presented at a time can be changed if the program is transfered to the RAM.

The flowchart of the DISPLAY routine is presented in Figures 3.19a through 3.19e . The blocks can be described as follows. the memory pointer index IY is loaded with the first data address, 1600H. N is loaded with the number of rows (or loops) to be displayed (224) and N=0 indicates the end of memory. NS controls the number of rows to be displayed at a time. After the initialization process the loop is started by loading the first data (the position error of the first loop) into the register pair HL. The data is checked. If it is positive a space is displayed on the screen and if it is negative a minus sign is displayed. The data is saved in a variable called NUMBER and the register pair DE is loaded with the decimal 10. The data (in HL) is divided by 10 (in DE) and the quotient (in HL) is saved in the variable QUOT.

The quotient (in HL) is multiplied by 10 (in DE). The result is transferred to DE and the data that was saved in NUMBER is loaded into HL. So, recalling that the subtraction does:

$$HL = HL\,(\text{previous}) - DE$$

and after this operation HL will be equal to the least significant bit of the decimal data. This value is saved in the variable ONES. Thus,

$$ONES = \text{dividend (data)} - 10\ QUOT$$

The LONGDIV subroutine ,presented in Figure 3.20 uses the same procedure mentioned above with the actual quotient being the dividend of the next division. The new remainder will be the second decimal digit (saved in the variable TENS). The process is repeated to get the next digits, saved in HUNDREDS, THOUS and TTHOUS. The biggest possible decimal number is 32767.

The decimal digits are converted to ASCII and displayed on the screen using the monitor subroutine ASCONV (refer to Appendix C). After the presentation of the least significant digit, register B is loaded with the number two and the monitor subroutine SPACES is called to display two spaces between the number just shown and the next.

Every time a conversion is completed the pointer is checked to verify if the row ended. At the end of the first row, for instance, the pointer (IY) will be 1610H, that is, the rightmost four digits are zero. The same happens every time a row is completed since eight two byte variables make a complete row. Thus, if this situation is detected, meaning that a row is completed, the monitor subroutine SCRLF is called to provide carriage return and line feed. Otherwise, a new data is converted. The loop continues until the end of the row.

Whenever a row is completed the end of memory must be checked. As mentioned earlier the number of rows left is controlled by the variable N. So, each time a row is completed, N is decremented and checked. If it is zero the program is ended, otherwise it continues.

The program is set up to present 10 rows at a time. The variable NS counts the number of rows and while NS is different of 10 the loop continues. When NS reaches 10 the program polls the keyboard input. While there is no input the program keeps polling the keyboard. When any key is typed, NS is set to zero and the loop continues.

8. The Subroutine TRANSFER

This subroutine is used to transfer the program MODEL from the EPROM to the RAM in order to debug the program. The starting address in the RAM is 1000H. The procedure to use this feature will be discussed in Appendix C. The subroutine is very simple and can be understood from the program.

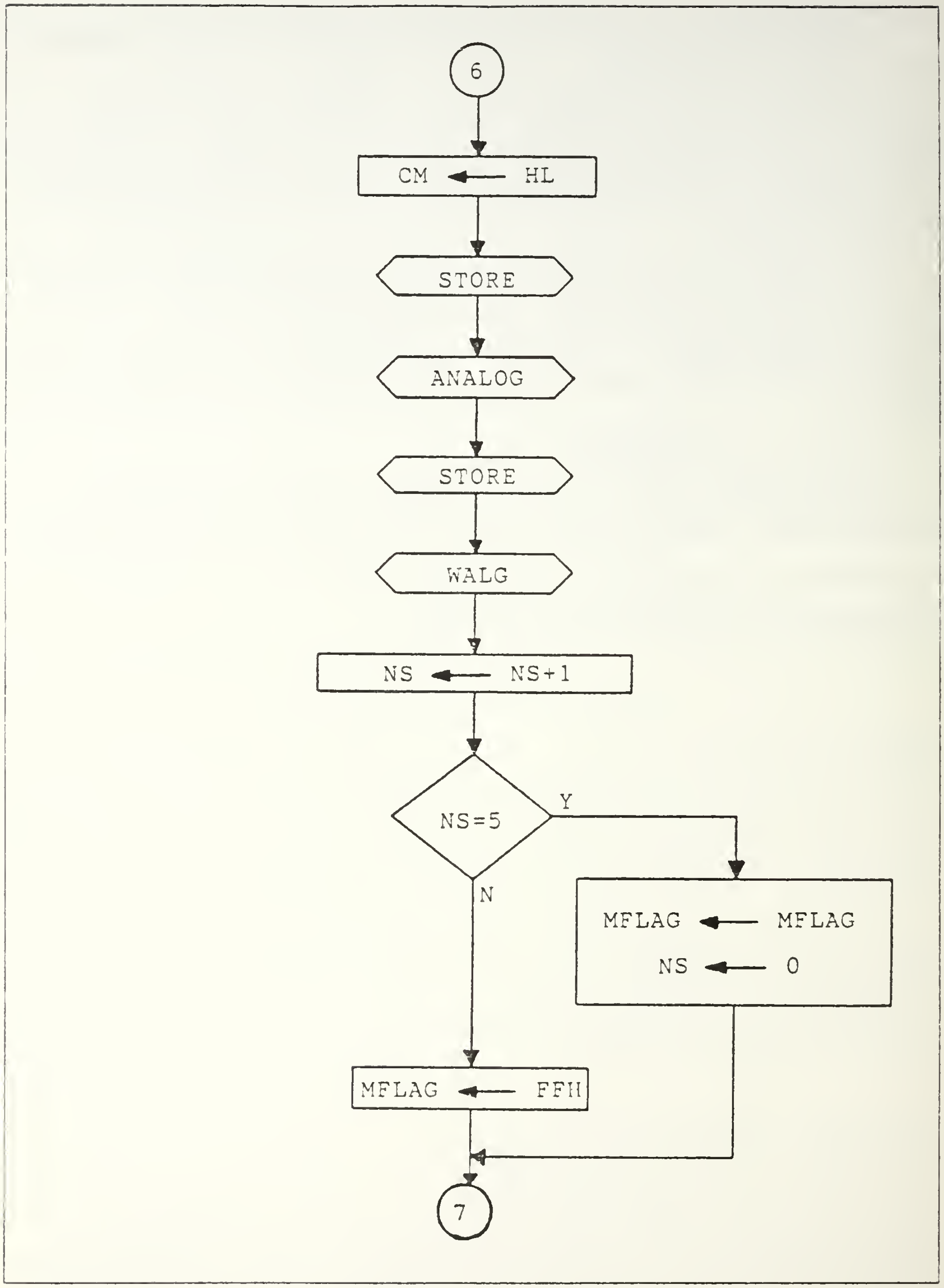

Figure 3.10 The Main Program - Flowchart G.

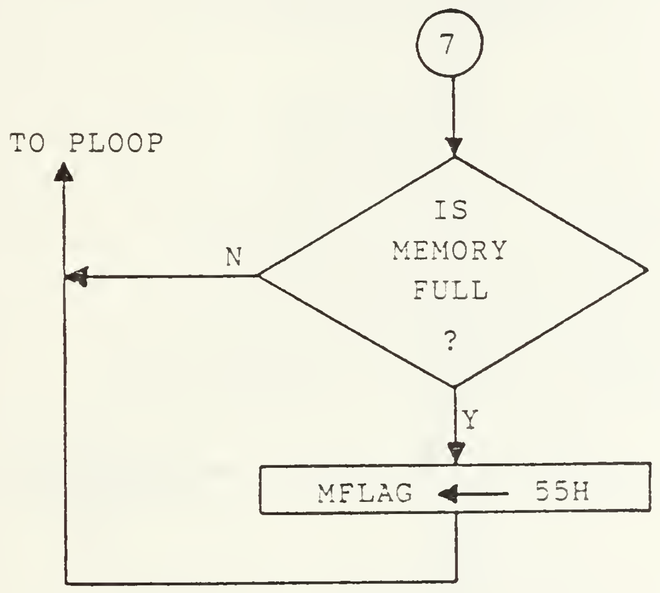

Figure 3.11 The Main Program - Flowchart H.

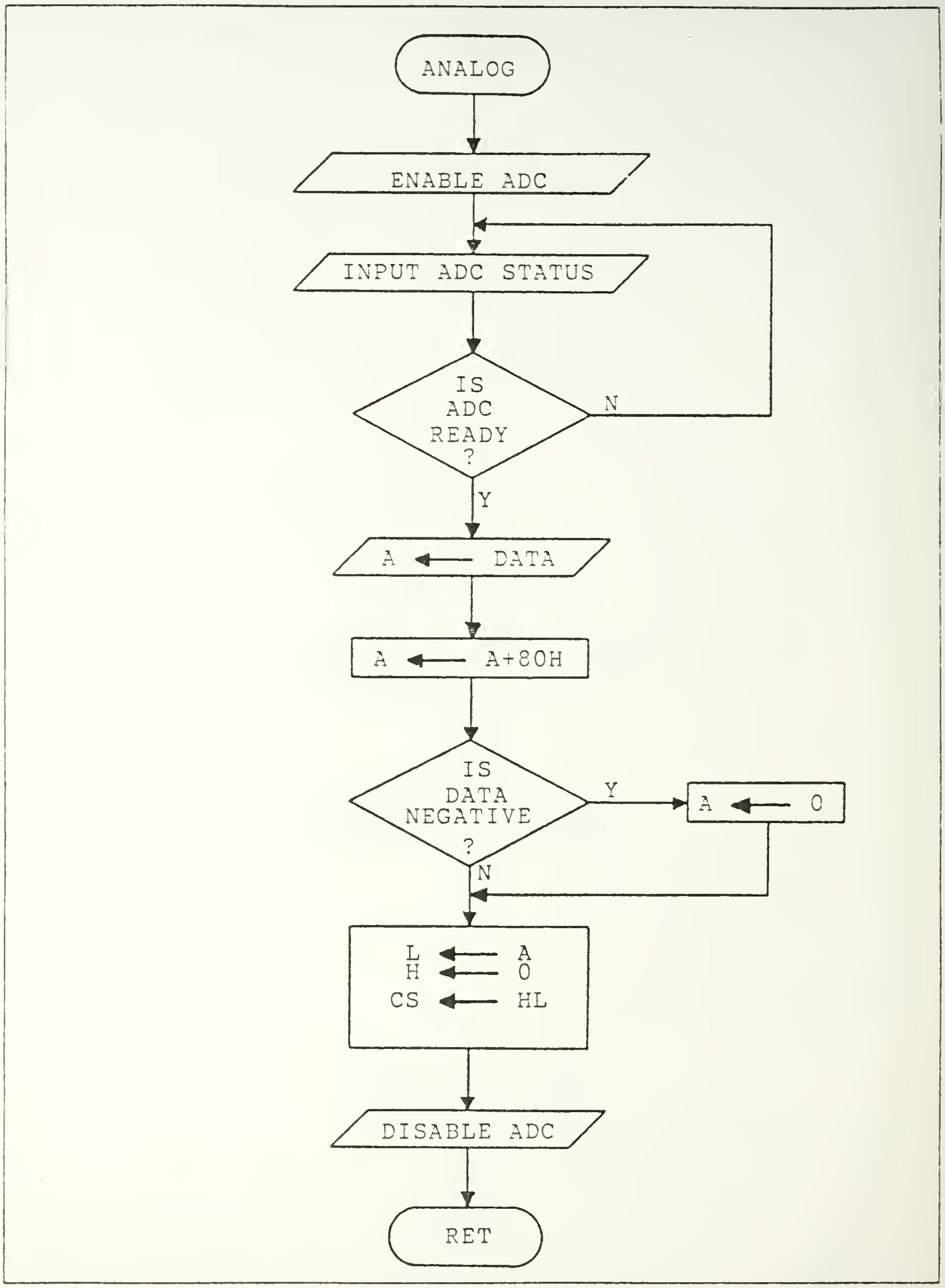

Figure 3.12 The subroutine Analog.

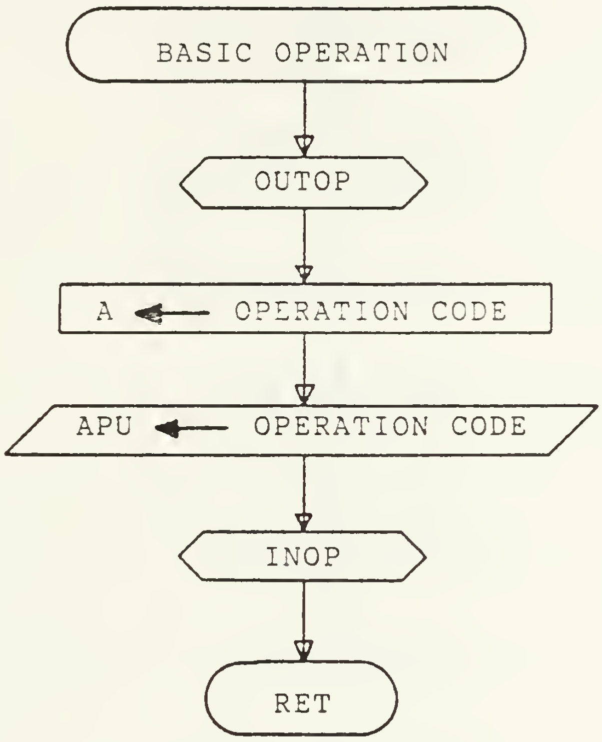

Figure 3.13 The Subroutine of Basic Operations.

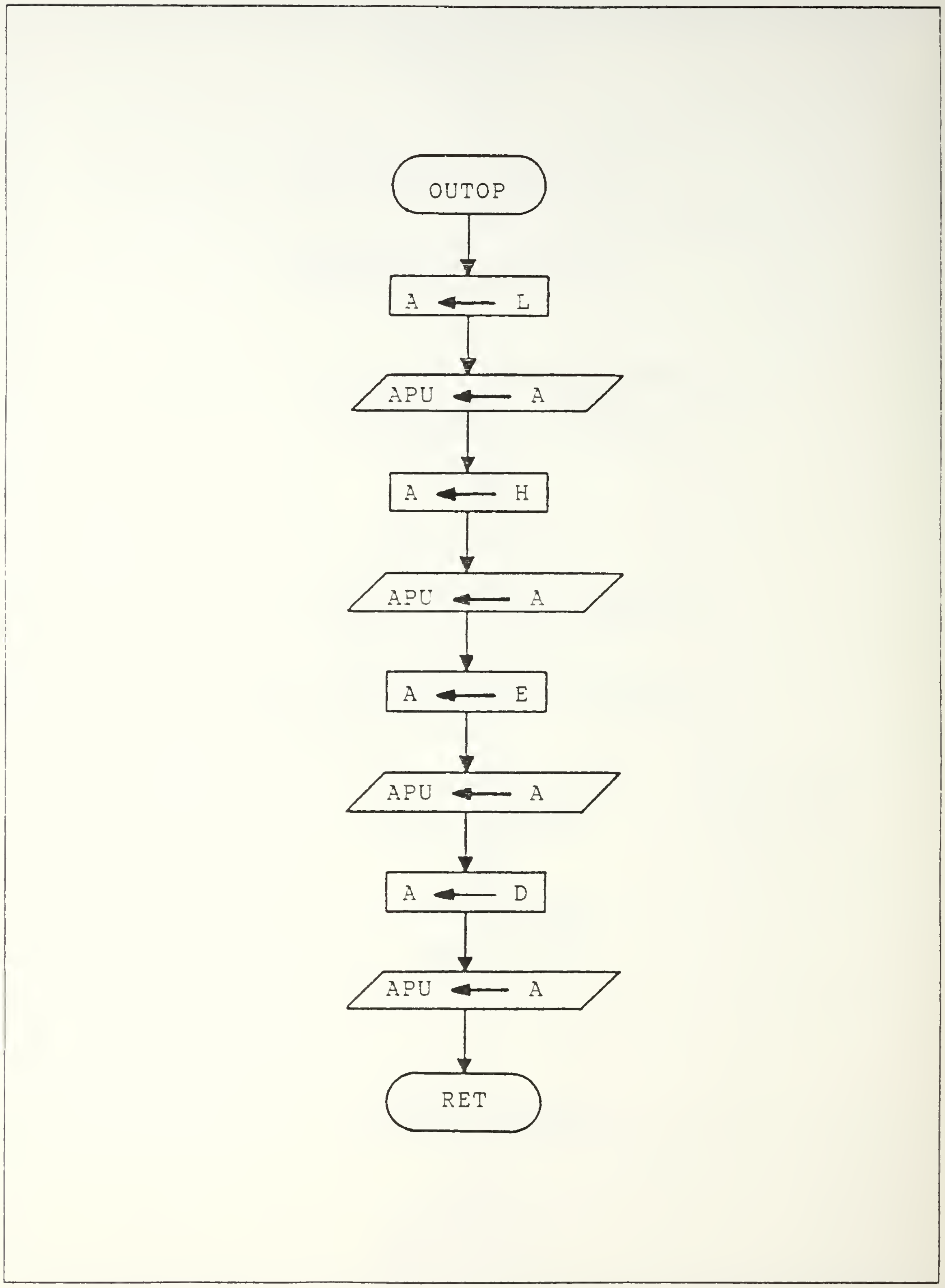

Figure 3.14 The Subroutine Outop.

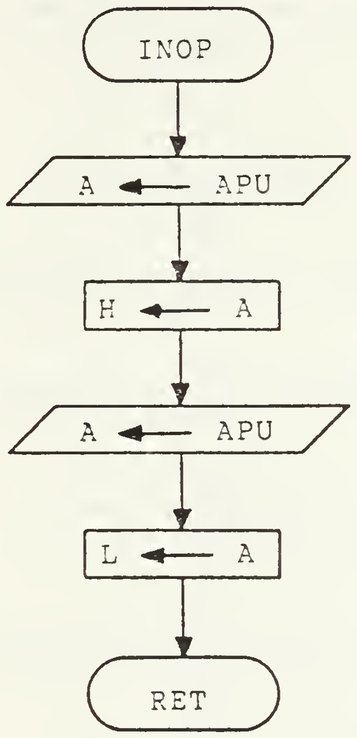

Figure 3.15 The Subroutine Inop.

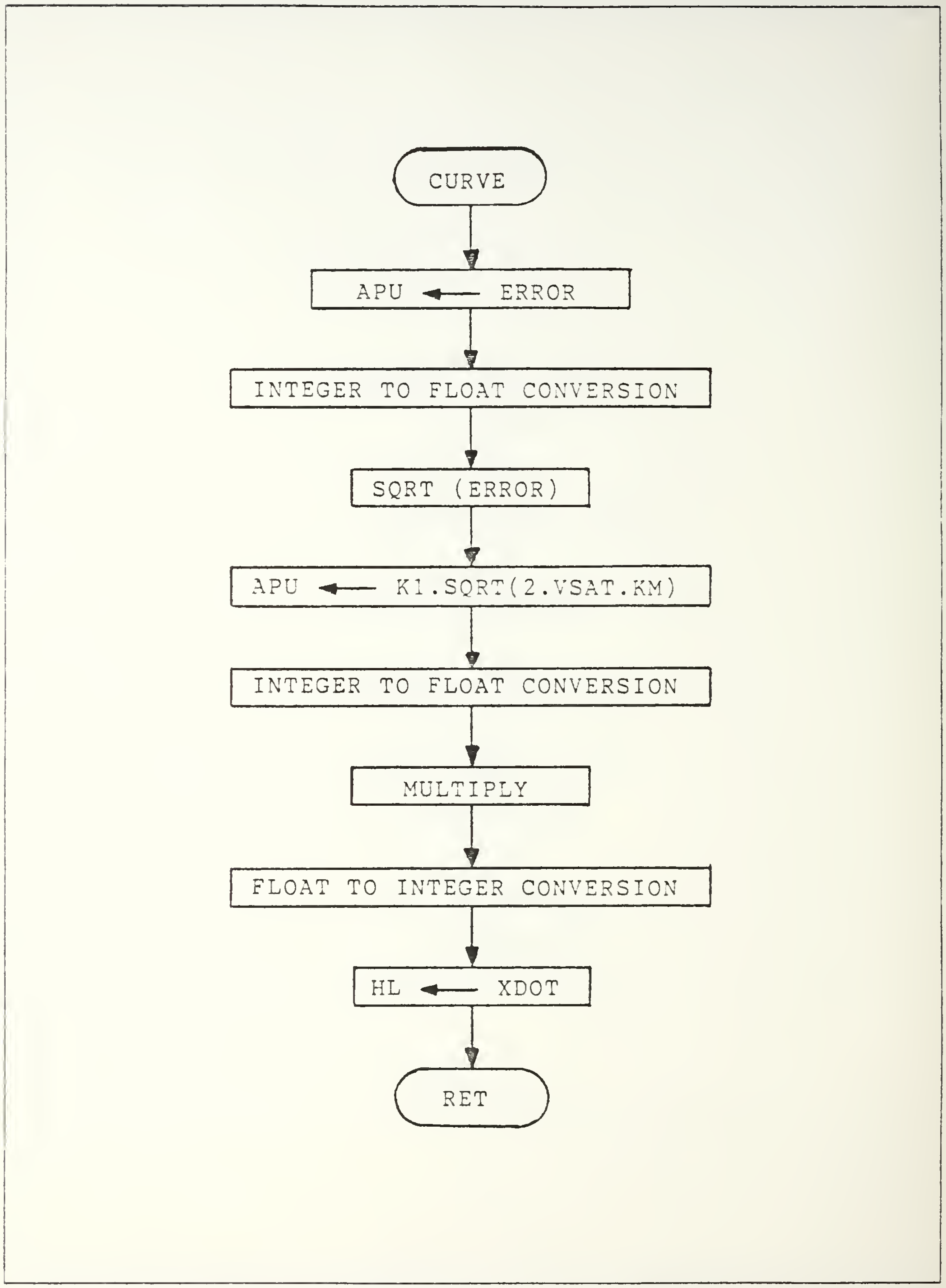
CURVE
APU ← ERROR
INTEGER TO FLOAT CONVERSION
SQRT (ERROR)
APU ← K1.SQRT(2.VSAT.KM)
INTEGER TO FLOAT CONVERSION
MULTIPLY
FLOAT TO INTEGER CONVERSION
HL ← XDOT
RET

Figure 3.16 The Subroutine Curve.

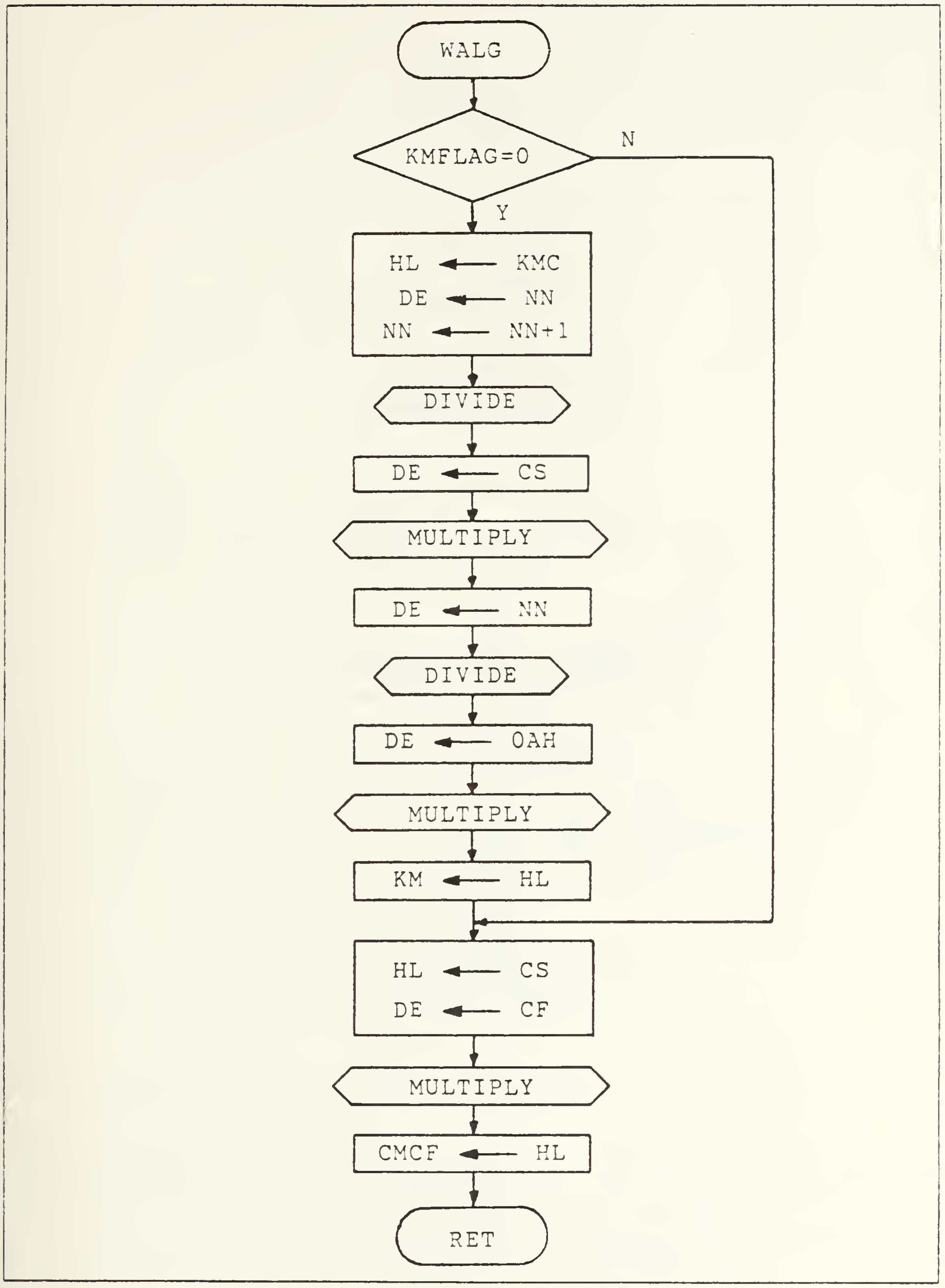

Figure 3.17 The Subroutine Walg.

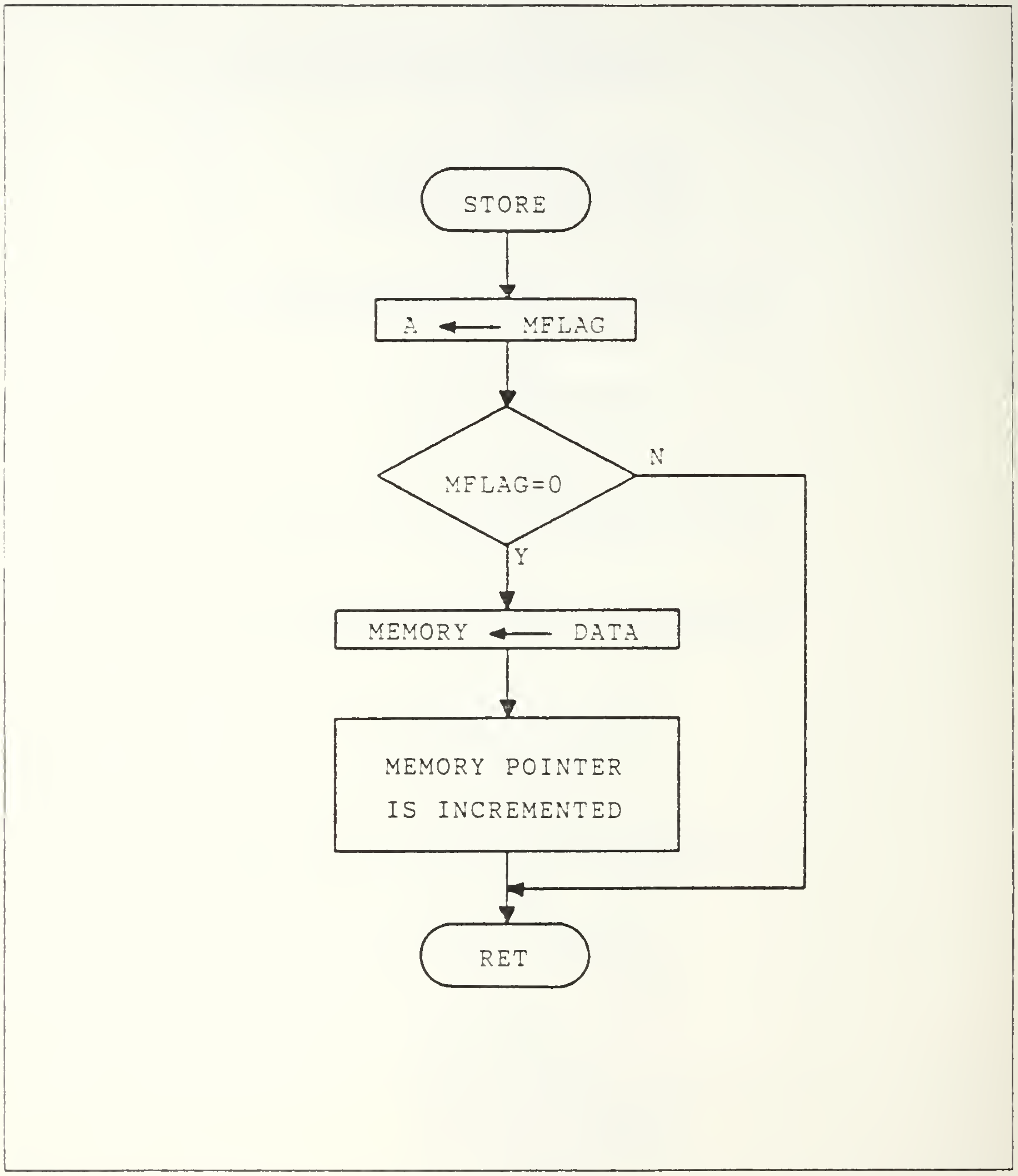

Figure 3.18 The Subroutine Store.

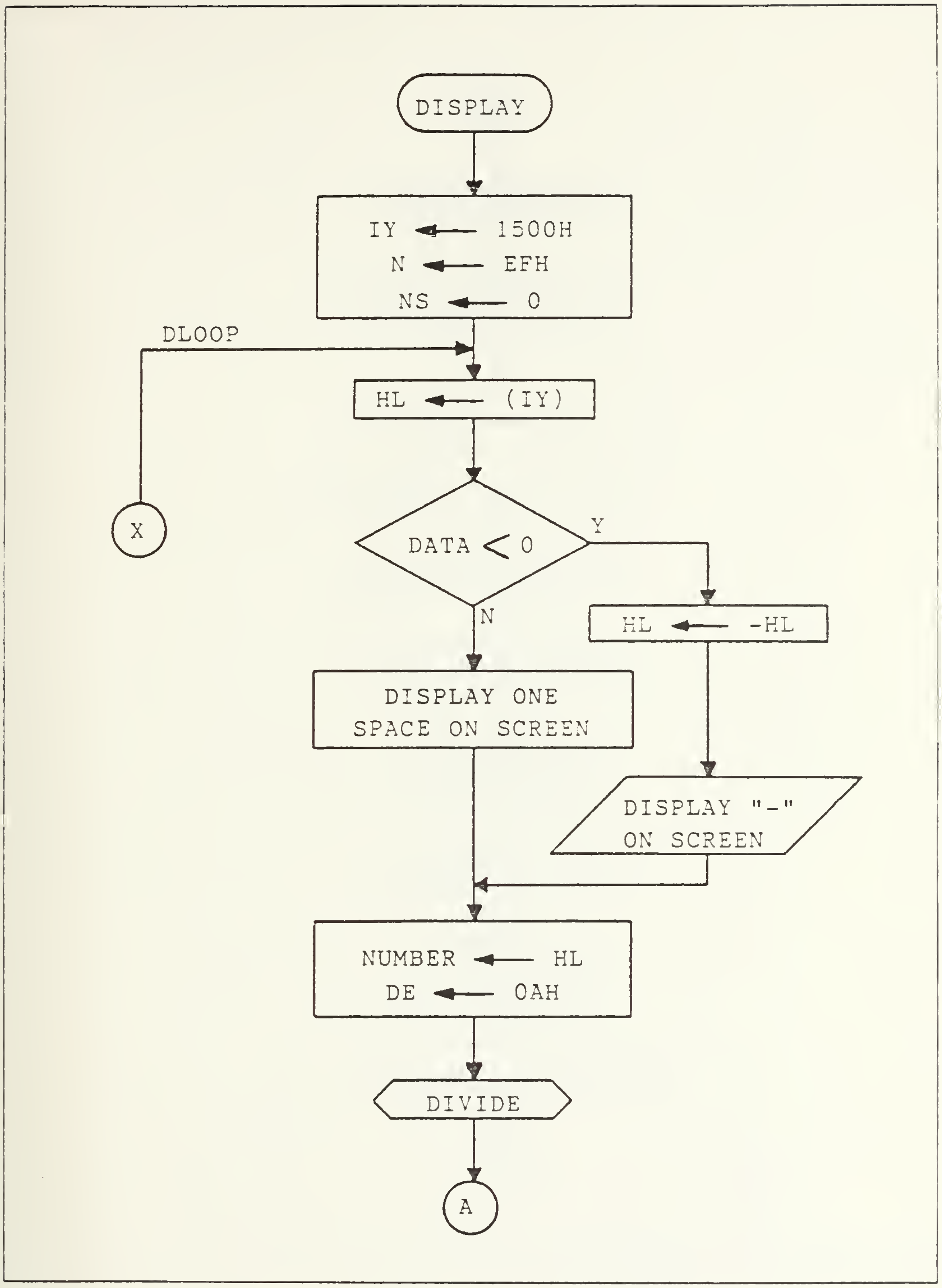

Figure 3.19a The Subroutine Display.

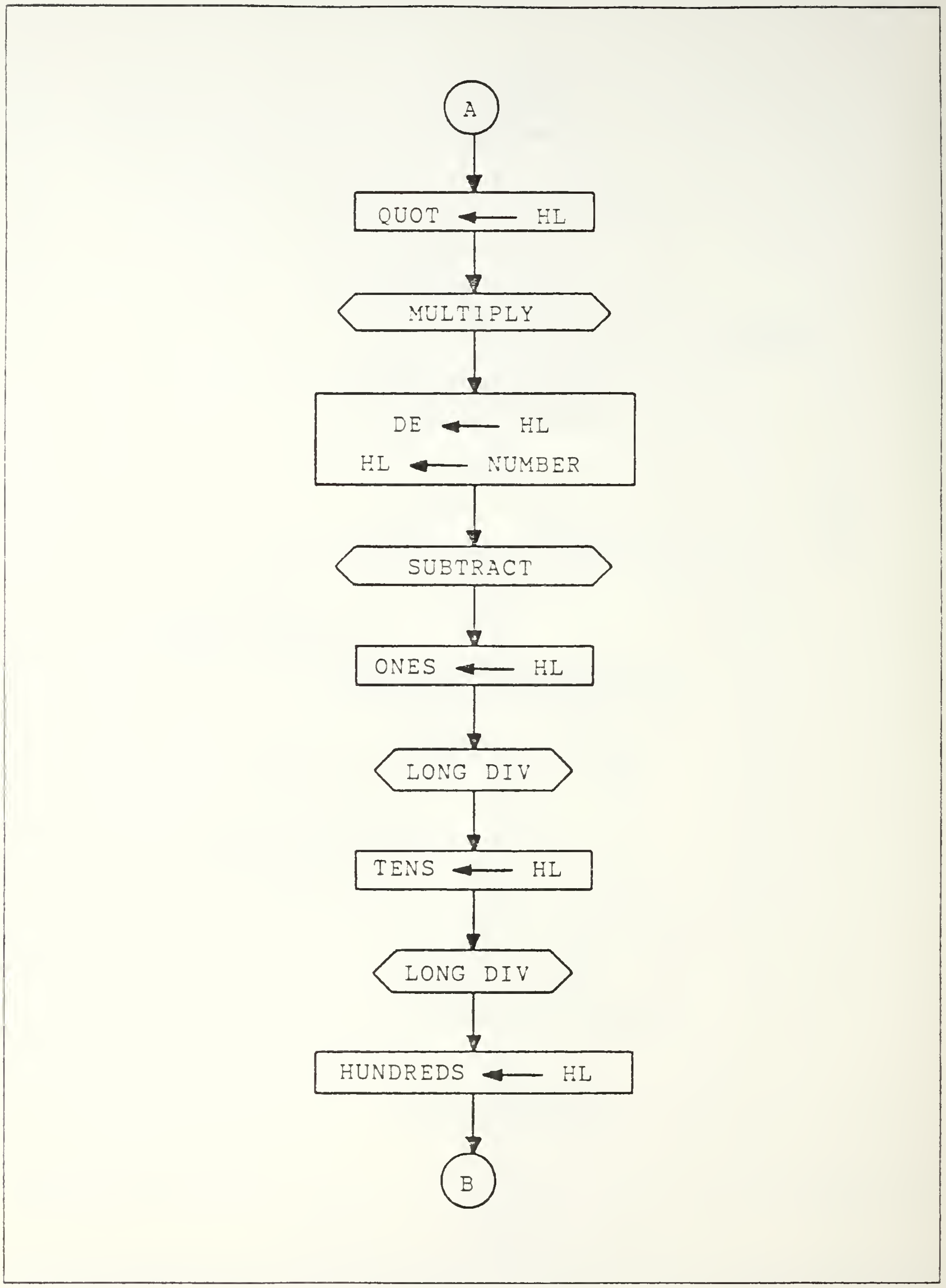

Figure 3.19b The Subroutine Display.

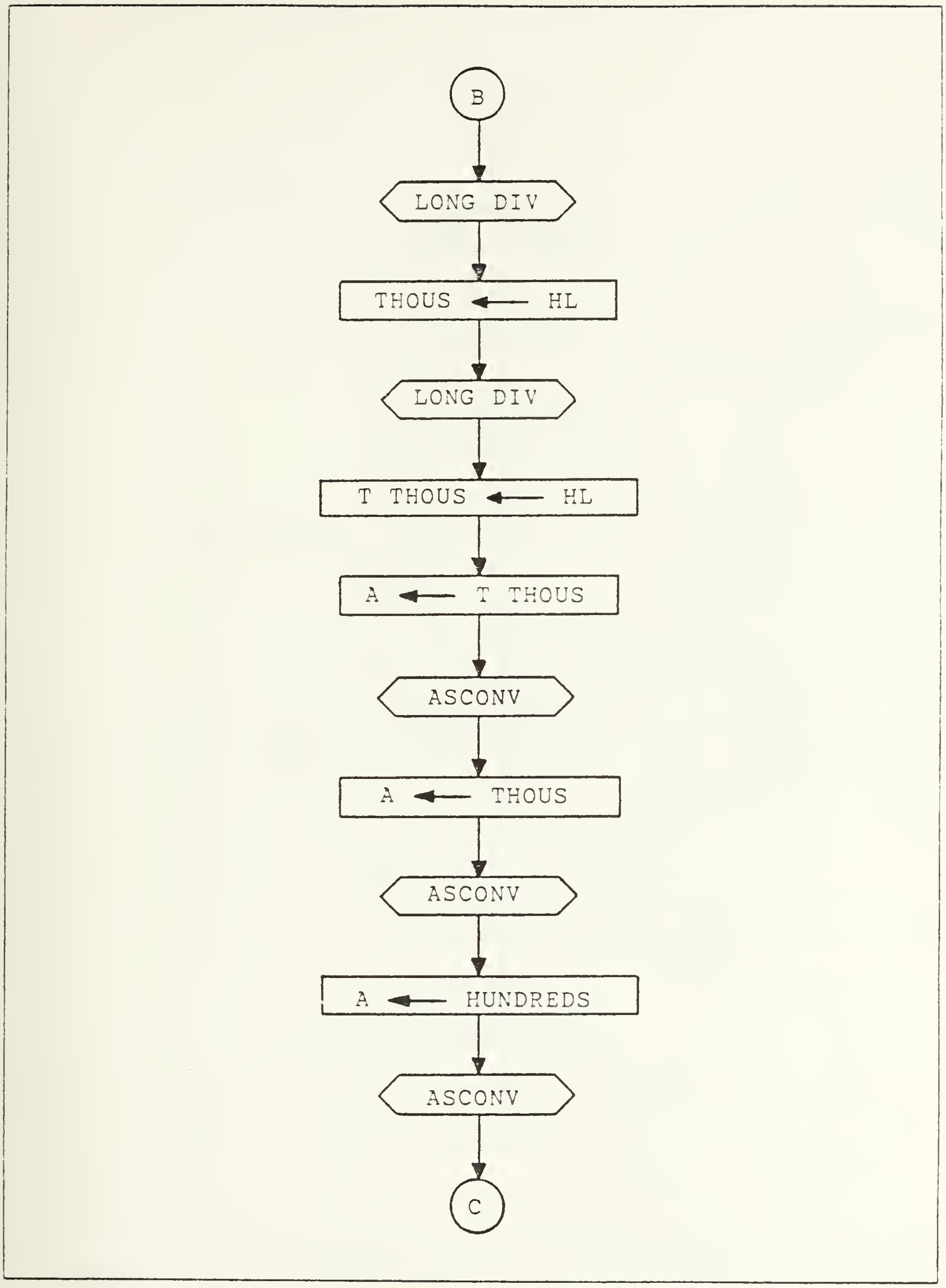

Figure 3.19c The subroutine Display.

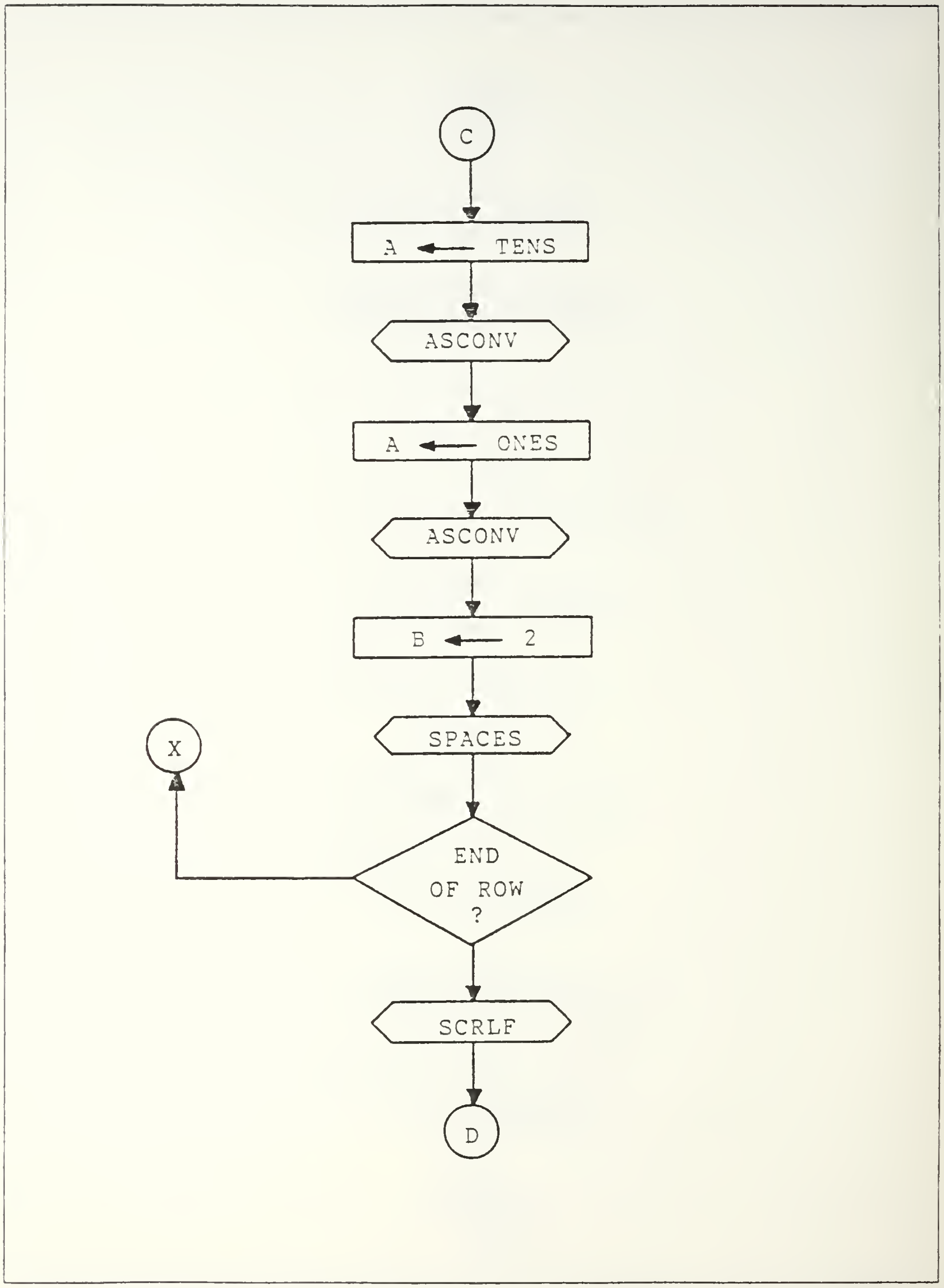

Figure 3.19d The subroutine Display.

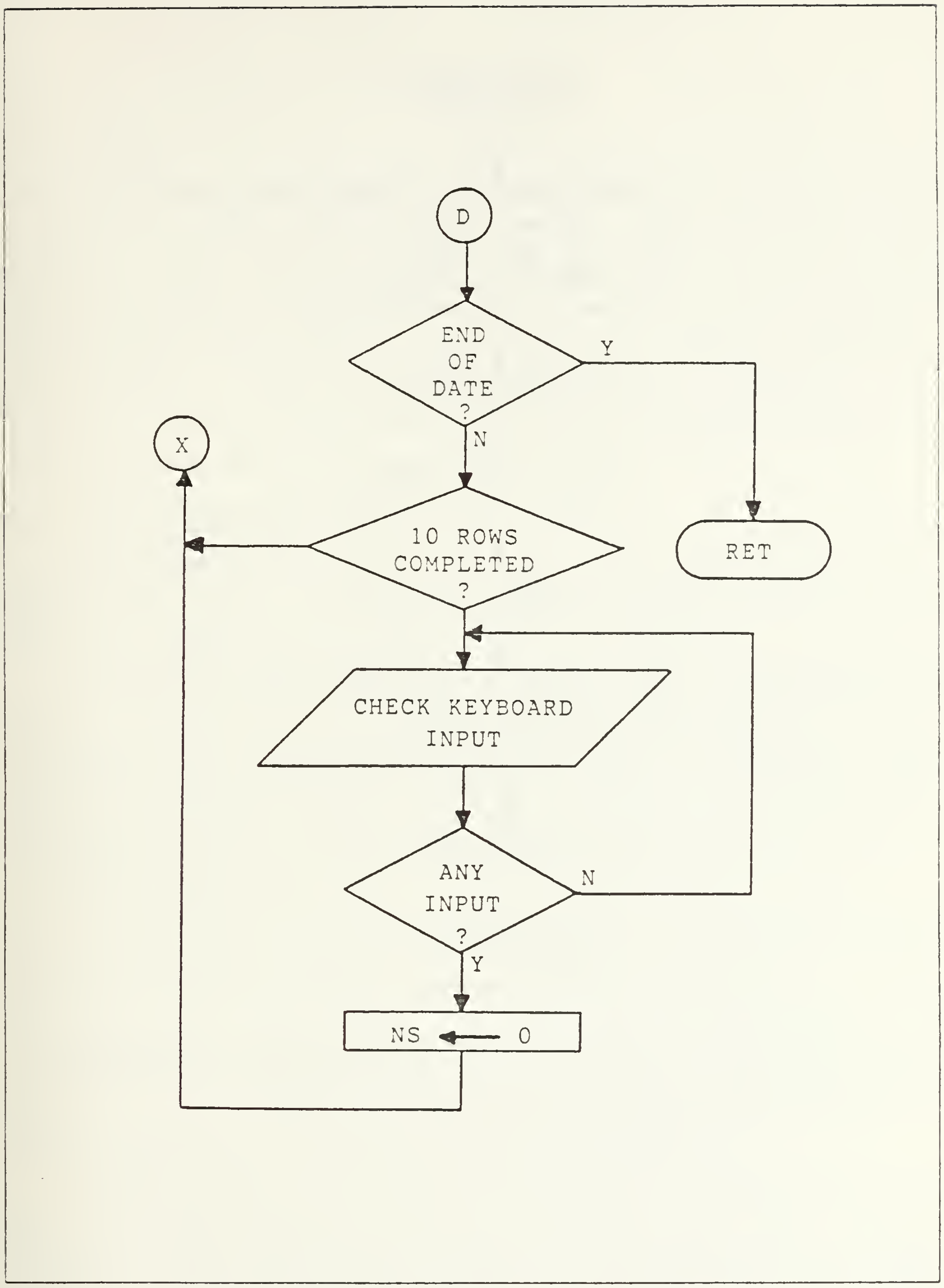

Figure 3.19e The subroutine Display.

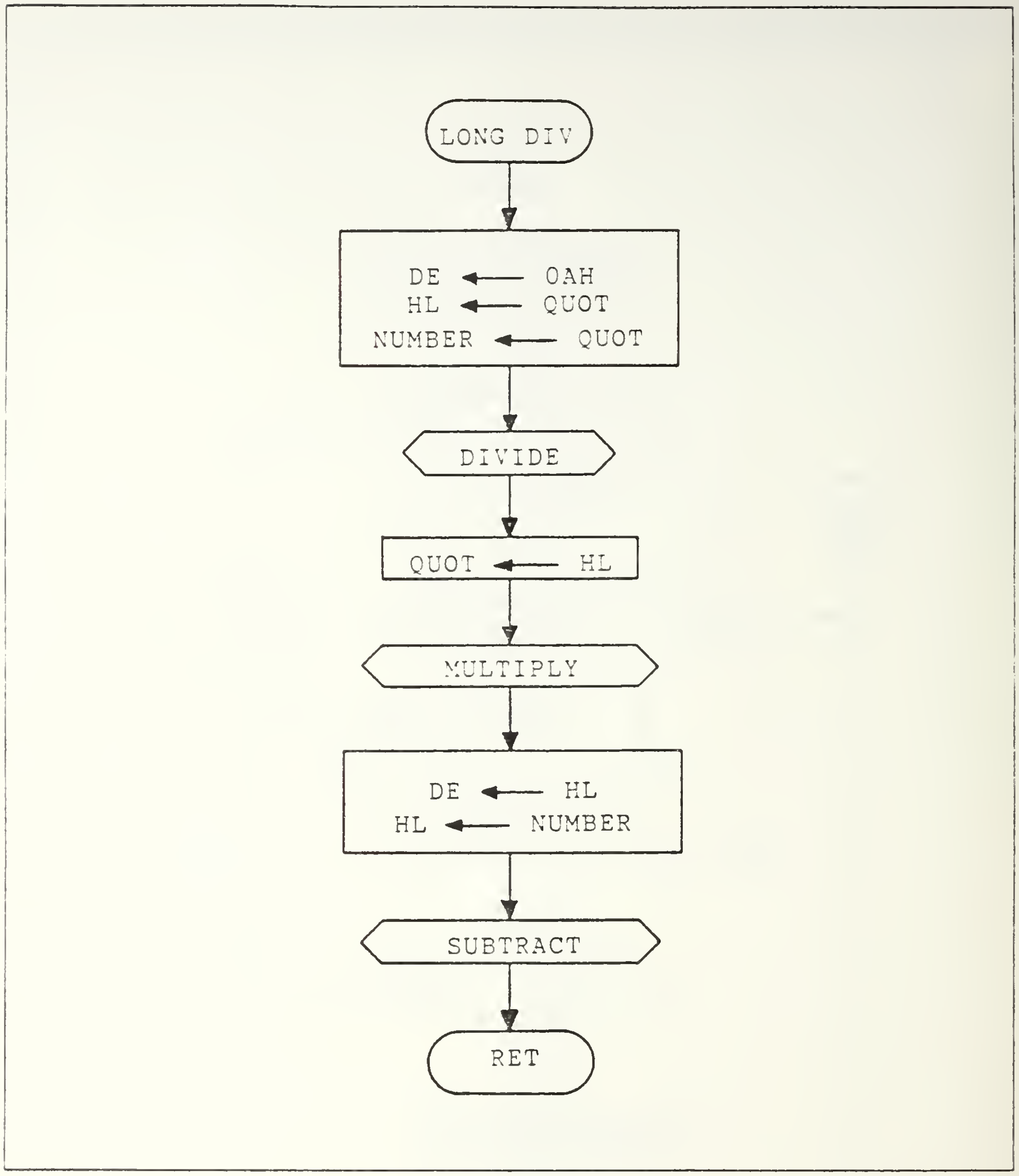

Figure 3.20 The Subroutine LONGDIV.

IV. HARDWARE

A. GENERAL

The hardware consists basically of two major parts, the microprocessor and the analog plant. Interconnecting them we have the digital to analog and analog to digital interfaces. A video terminal is coupled to the microprocessor and all inputs and outputs are accessed by using this facility. An oscilloscope connected to the output of the plant measures the analog output. This voltage is also available in a digital form using (off-line) the routine Display. The whole hardware is mounted in three protoboards, one for the microprocessor, one for the analog plant and a third one for the extra memories used to store data.

B. THE MICROPROCESSOR

The microprocessor is a general term used to refer to the digital system that implements the model, controller and the identification algorithm. The circuit that shows all the digital system and interfaces is presented in Figure 4.1 .

The central processing unit is the Z-80, the memories are two EPROMS (2716) and five RAM's (MK 4118). The memory and interface decoders are three 74LS138. The parallel interface is the M8255 and the serial interface is the MC68661. Besides the mentioned chips that are normally found in microprocessors there is the Arithmetic Processing Unit (APU), Intel 8231, that performs all the mathematical operations needed in the digital system.

The organization of the memories is shown in Table 1 . The monitor and model (including the controller and algorithm) are loaded into the EPROM's. The RAM addresses from 1000H to 13A0H are used for scratch. This is an important point in the development of the system because it allows the whole program "Model" to be transfered from a non-erasable memory to an erasable one. Once the program is transfered it can be modified and run in the RAM using the monitor features for debugging purposes. This procedure will be detailed in Appendix C.

The RAM addresses from 13A0H to 1600H are used to store the data segment, that is, all variables defined in the monitor and program Model. Also, every byte originated by typing a key in the keyboard goes to a particular stack in this memory area. The microprocessor stack pointer is initialized at 1600H.

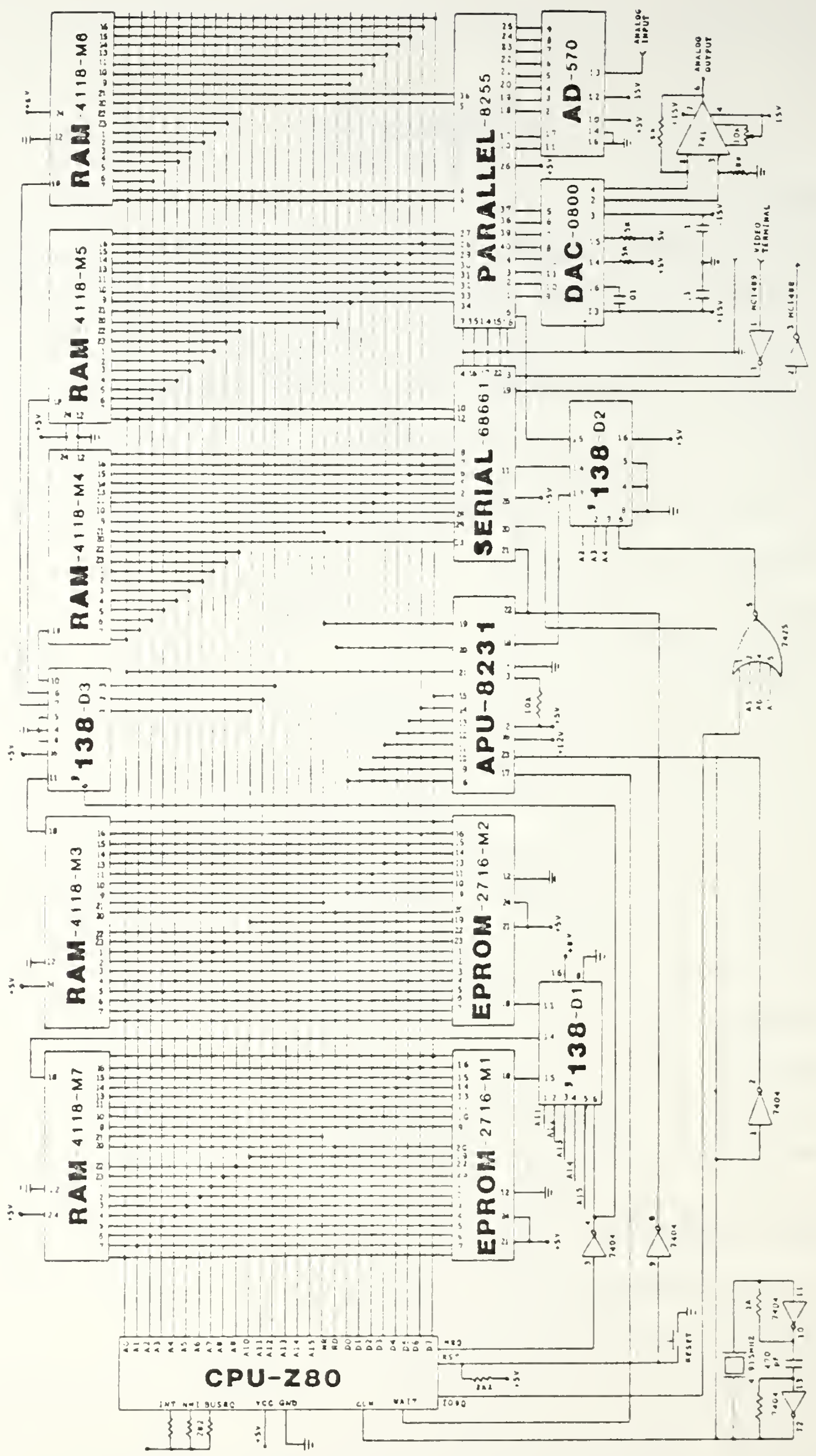

Figure 4.1 The Microprocessor and Interfaces.

TABLE 1
MEMORY ORGANIZATION

#	START	END	TYPE	FUNCTION
M1	0000	07FF	EPROM 2716	monitor + model
M2	0800	0FFF	EPROM 2716	model (cont.)
M3	1000	13FF	RAM MK4118	scratch/data seg
M4	1400	17FF	RAM MK4118	data seg/storage
M5	1800	1BFF	RAM MK4118	data storage
M6	1C00	1FFF	RAM MK4118	data storage
M7	2000	23FF	RAM MK4118	data storage

The memory from 1600H to 23FFH is the so called "data memory". It stores the intermediate results of program Model for future presentation on the screen. After a run the subroutine Display can be invoked to show these data. The RAM memories M4, M5, M6 and M7 (refer to Figure 4.1), used for data storage, were created to allow the off-line display of the mentioned data. When the digital servo was first tested (without the analog plant) it used on-line routines to show the results on the screen. However, each time the routine was called to present one variable it spent more than 400 microseconds. This would consist in a major problem for incorporating the analog plant. So, the extra memories were added to the system in order to provide room for the massive data record generated by the subroutine Store.

As shown in the circuit, the decoder D1 addresses the EPROM's M1 and M2 and the RAM M7. The decoder D2 addresses the serial and parallel interfaces and the APU. Decoder D3 addresses the data memories M3, M4, M5 and M6.

Decoder D2 is selected by addresses A2, A3 and A4 and its outputs Y0 (pin 15), Y1 (pin 14) and Y2 (pin 13) are the chip select commands for the 8255 (parallel port), 68661 (serial port) and APU, respectively. So, depending on the addresses A1 and A0, the parallel interface will be at addresses 00, 01, 02 and 03 and the serial interface will be at addresses 04, 05, 06 and 07. The APU will be at addresses 08, 09, 0A and 0B.

C. THE ARITHMETIC PROCESSING UNIT

This integrated circuit deserves a special attention in this research. First of all, it played a very important role on the software development in taking over all of the mathematical operations. Secondly, it is a preliminary chip and its application consists in a parallel research.

The Arithmetic Processing Unit (APU) 8231, as referred in [Ref. 5] has the following features:

1. Fixed point, single and double precision (16/32 bits).
2. Floating point single precision (32 bits).
3. Binary data formats.
4. Add, subtract, multiply and divide.
5. Trigonometric and inverse trigonometric functions.
6. Square roots, logarithms, exponentiation.
7. Float to fix and fix to float conversions.
8. Stack oriented operand storage.
9. Direct memory access or programmed I/O data transfer.
10. General purpose 8 bits data bus interface.

In the 16 bit program this chip is used to perform the following operations: addition, subtraction, multiplication, division, square root, fix to floating point conversion and vice versa. In general the operations were done with fixed point operands to keep the program working with 16 bit variables. However, some operations where the APU stack could be used as a temporary register, were done in 32 bit floating point. This happened in the subroutine CURVE and in the trapezoidal integration in the main program. The advantage of this method is that the accuracy of the computations is increased without defining 32 bit variables.

The 16 bit format is straight forward. It works with binary operands represented in two's complement values. The sign of the operand is located in the most significant bit (at the leftmost position). Positive values are represented by zero and negative values are represented by one. This format can represent numbers in the range from -32768 to 32767.

The 32 bit floating point format permits us to represent positive and negative numbers from 2.7×10^{-20} to 9.2×10^{18} and zero. As depicted in Figure 4.2, the 32 bit numbers consist of four parts: mantissa, exponent, exponent signal and mantissa signal. The mantissa uses the 24 rightmost bits (0-23), the exponent uses the next six bits (24-29), the exponent sign uses the next bit (bit 30) and the mantissa sign is represented at the leftmost bit (bit 31).

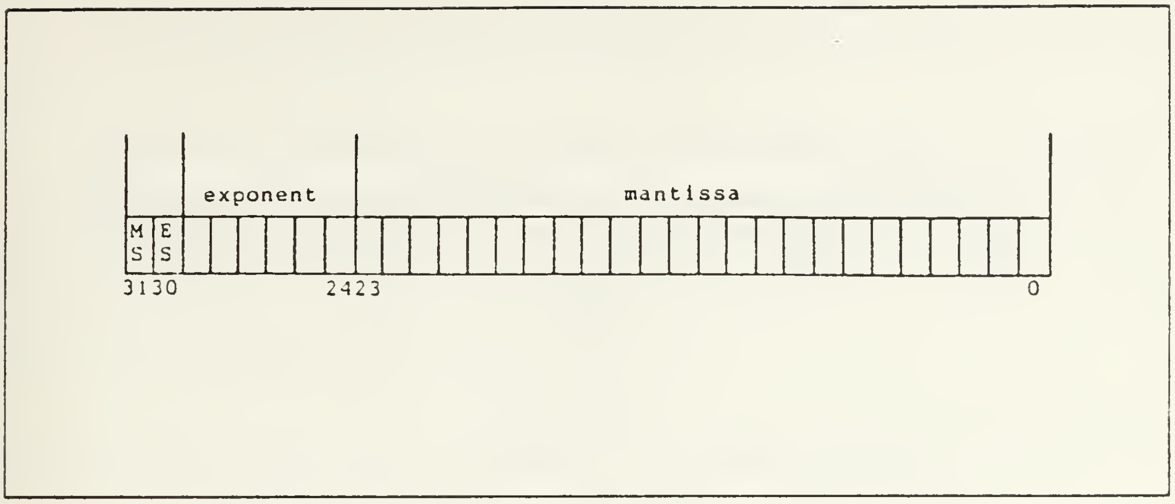

Figure 4.2 32 Bit Floating Point Register.

A requirement of this processor is that the data, when using 32 bit floating point format, be represented by a fractional mantissa value between 0.5 and 1.0 multiplied by two raised to an appropriate power, that is, value = mantissa $\times$ 2^{exponent} .

Illustrating the explanation above with an example, let's take the number 624 and convert it to 32 bit floating point:

$$624 = 512 + 64 + 32 + 16 = 2^9 + 2^6 + 2^5 + 2^4 =$$
$$= 0.1 \times 2^{10} + 0.0001 \times 2^{10} + 0.00001 \times 2^{10} + 0.000001 \times 2^{10} =$$
$$= 2^{10} \times (0.100111)$$

The binary representation of the above example is shown in Figure 4.3 . The hexadecimal representation of this four byte number is 0A9C0000H. This is the code to be sent to the APU in order to create the floating point equivalent to the decimal 624.

The APU uses a stack to store the operands and results. It is an eight level 16 bit wide data stack, as shown in Figure 4.4 . The same stack is used to deal with 32 bits but, in this case, the configuration changes to a four level stack. The upper level is called TOS (top of stack) and the level below the TOS is called NOS (next on stack).

Data are written onto the stack, eight bits at a time in the order A1, A2, A3, etc. and are removed in the reverse order. For instance, suppose the operation B $-$A = C, where B = B2 B1, A = A2 A1 and C = C2 C1. In a subtraction the operand in the TOS is subtracted from the operand in the NOS and the result is stored onto the TOS. Thus, the bytes must be sent in the following order: B1, B2, A1, A2. The result is

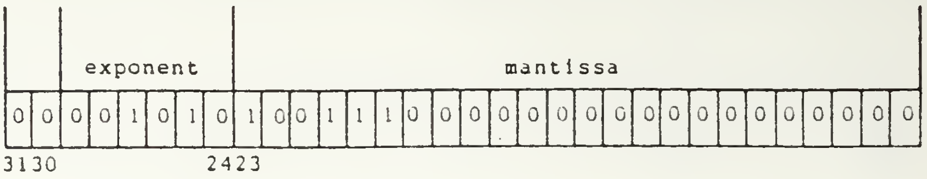

Figure 4.3 Example of a 32 Bit Floating Point Number.

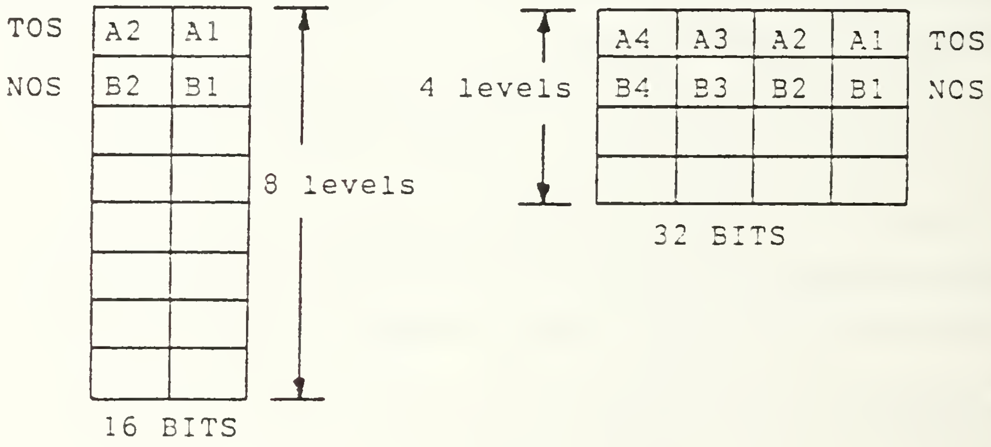

Figure 4.4 Stack Configuration for APU 8231.

retrieved in the order C2, C1. Considering the way the software was designed, the operand B would be onto the register pair HL and the operand A onto the register pair

DE. The result would be in HL. So, the data from the registers would be sent to the APU in the following order: L. H. E. D and the result would be received in the order H, L.

The data entry and data removal process is illustrated in Table 2 . Data entry is accomplished by bringing the chip select (CS), the command/data line (A0) and the write line (WR) low. A new entry occupies the TOS, pushing the previous TOS to NOS. Data removal is performed by setting CS, A0 and RD low. The data in TOS is removed and the data in NOS is moved to TOS.

TABLE 2

DATA ENTRY AND DATA REMOVAL PROCESS

WR	RD	CS	A0	OPERATION	INSTRUCTION
1	0	0	0	Read	IN A,(08H)
0	1	0	0	Write	OUT (08H),A
0	1	0	1	Command	OUT (09H),A
1	0	0	1	Read Status	IN A,(09H)

After the data have been entered the required operation can be performed by issuing a command. The command operation is accomplished by bringing the chip select line low, command/data line high and write line low.

It can be seen in Figure 4.1 that the READY line of the APU (pin 17) is connected to the WAIT line of the CPU (Z-80). The READY line is normally high and is pulled low by the APU when certain conditions occur. Basically, the READY line goes low when the APU is busy and either data or command operation is requested. If this happens, the WAIT line of the CPU goes low and it waits. When the operation is completed the READY line goes high and the result is available at the TOS. Then the CPU can retrieve the data from the APU. The process of removing data is illustrated in the first row of Table 2 .

The software for the the APU is quite simple and can be easily understood from the program. However, in order to figure out how the operations work and to have a complete knowledge of the chip, some small routines were written to test individual operations. These routines were written in machine language using the features available in the monitor.

As pointed out earlier in this Chapter, the memory addresses from 13A0H to 1600H are used for the data segment, that is, to store the variables used in the program. But this section of memory is not completely filled and there is some space available. The test program can start at address 15A0H, for instance. The sequence to write the program is the following:

1. Turn on the system (video terminals and power supplies).
2. Reset the system by pressing the reset switch.
3. In the keyboard, type: C15A0,xxxx <ret>. This will be the segment of memory to be used (xxxx is the end address of the small program).
4. Type in the machine language program, entering one byte at a time, that is, type two hexadecimal numbers and hit the return key.

The procedure to run the routines and get the results is as follows:

1. Type G15A0,yyyy <ret>, where yyyy is the end address plus one of the routine, that is, yyyy is equal to xxxx + 1.
2. When the execution is completed the registers will be automatically displayed on the screen and the result of the operation under test will be shown in the register HL.

1. Subtraction Routine

The APU performs NOS − TOS and stores the result onto the TOS. The result is removed from the TOS and stored into the register pair HL, where it can be checked after the execution of the routine. In the example shown in Table 3 the operands are 0002 (TOS) and 0007 (NOS). So, the expected result is 0005. Other examples can be done by just changing the operands. In this particular example the last address is 15B9H and the execution command should be: G15A0,15BA <ret>. With this command the program will run and the CPU registers will be displayed on the screen.

2. Multiply Routine

The APU performs NOS × TOS and stores the result onto the TOS. The result is removed from TOS and stored in HL where it can be checked after the execution. The routine is presented in Table 4 using the operands: FFFAH = −6 and 0001H = 1. Thus, the expected result is FFFAH = −6 . Other operands were used to check the routine: FFFAH = −6 as first and second operand with result 0024H = 36 and FFFAH = −6 and 0002H = 2 with result FFF4H = −12.

3. Square Root Routine

This routine is performed using 32 bit (floating point) format. Therefore, the 16 bit integer operands must be converted to floating point and the result, converted

TABLE 3

SUBTRACTION ROUTINE FOR TESTING THE APU

Assembly Language	Address	Machine Language	Comments
LD A,07H	15A0	3E07	APU <-- first operand
OUT (08H),A	15A2	D308	
LD A,00H	15A4	3E00	
OUT (08H),A	15A6	D308	
LD A,02H	15A8	3E02	APU <-- second operand
OUT (08H),A	15AA	D308	
LD A,00H	15AC	3E00	
OUT (08H),A	15AE	D308	
LD A,6DH	15B0	3E6D	APU <-- subtraction
OUT (09H),A	15B2	D309	
IN A,(08H)	15B4	DB08	HL <-- result
LD H,A	15B6	67	
IN A,(08H)	15B7	DB08	
LD L,A	15B9	6F	

back to integer. In the example shown in Table 5 the operand is 0270H = 624 and the expected result is 18H = 24, that is, the square root is rounded off to the next less integer. As in the previous routines the result can be checked at the HL register after execution. Since the start address is 15A0H and the end address is 15B9H, the execution command will be : G15A0,15BA <ret>.

D. THE INTERFACES

The serial and parallel interfaces, MC68661 and M8255, respectively, are largely used with eight bit microprocessors and will have just a short explanation. The serial interface interchanges information with the video terminal. So, all the commands coming from the terminal keyboard and all information going to the screen are formatted by this chip. The parallel interface connects the microprocessor with the analog plant. So, the plant input (V) and the plant output (CS) pass through this interface, in digital format. As shown in Figure 4.1 the three ports available in the 8255 are used in the following way:

TABLE 4

MULTIPLICATION ROUTINE FOR TESTING THE APU

Assembly Language	Address	Machine Language	Comments
LD A,FAH	15A0	3EFA	APU<--first operand
OUT (08H),A	15A2	D308	
LD A,FFH	15A4	3EFF	
OUT (08H),A	15A6	D308	
LD A,01H	15A8	3E01	APU<--second operand
OUT (08H),A	15AA	D308	
LD A,00H	15AC	3E00	
OUT (08H),A	15AE	D308	
LD A,6EH	15B0	3E6E	APU<--mult. command
OUT (09H),A	15B2	D309	
IN A,(08H)	15B4	DB08	HL<--result from APU
LD H,A	15B6	67	
IN A,(08H)	15B7	DB08	
LD L,A	15B9	6F	

1. Port A provides the digital output (V) to the digital to analog converter (DAC 0800)
2. Port B receives digital input (CS) from the analog to digital converter (AD570)
3. Port C is used to control the AD converter

The digital to analog converter (refer to Figure 4.5) is always converting the digital input to a continuous voltage output between pins 2 and 4. The reference voltages, applied at pins 14 and 15 are +5 V and −5 V, respectively. The resistors connected to these terminals are called reference resistors.

The full scale output current (I_{fs}), represented by the sum of the currents at pins 2 and 4, is related to the reference voltages and resistors in the following way:

$$I_{fs} = (+V_{ref}/R_{ref}) \times (255/256)$$

$$I_{fs} = I_0 + \bar{I}_0$$

$$V_{ref} = +5-(-5) = 10\ \mathrm{V} \rightarrow I_{fs} \sim 2\ \mathrm{mA}$$

TABLE 5

SQUARE ROOT ROUTINE FOR TESTING THE APU

Assembly Language	Address	Machine Language	Comments
LD A,70H	15A0	3E70	APU <-- operand
OUT (08H),A	15A2	D308	
LD A,02H	15A4	3E02	
OUT (08H),A	15A6	D308	
LD A,1DH	15A8	3E1D	int./float command
OUT (09H),A	15AA	D309	
LD A,01H	15AC	3E01	sq. root command
OUT (09H),A	15AE	D309	
LD A,1FH	15B0	3F	float/int. command
OUT (09H),A	15B2	D309	
IN A,(08H)	15B4	DB08	HL <-- result
LD H,A	15B6	67	
IN A,(08H)	15B7	DB08	
LD L,A	15B9	6F	

In full scale, that is, with all digital inputs equal to 1, the current I_0 is 2 mA and the current $\overline{I_0}$ is 0. So, to get +5 V at the operational amplifier output, R2 must be 2.5 KΩ . In zero scale, that is, with all TTL inputs equal to 0, $I_0 = 0$ and $I_0 = 2$ mA. Thus, to obtain −5 V at the output R1 must also be 2.5 KΩ . In the actual design, the plant is being driven by an input of ± 10 Volts and the resistors are both 5KΩ .

In the ADC, located at the plant output, the input can vary from − 5 V to +5 V, providing digital outputs from 00H to FFH, respectively. This is not compatible with the two's complement format of the microprocessor, where 00H corresponds to 0 and FFH corresponds to − 1. In order to correct this discrepancy, every time a number comes from the ADC, the program adds 80H. The effect of this correction is illustrated in Table 6 . Looking at the Table one can see that a difference of 1 inside the microprocessor corresponds to a difference of 0.039 volts in the analog plant (5 volts/2^7) because we are dealing with eight bit numbers. There is a difference of 80H between the second and third columns, that is, between the binary numbers at the

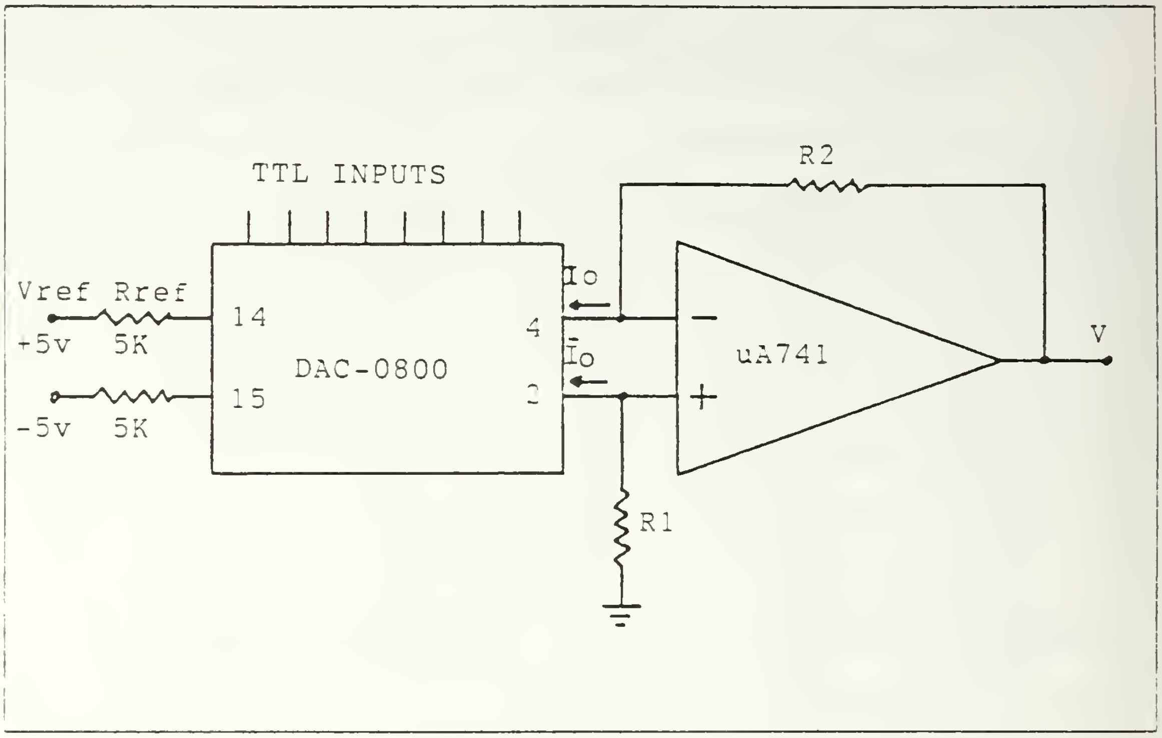

Figure 4.5 Digital to Analog Converter.

interfaces and the two's complement numbers inside the microprocessor. This is the reason why the software adds 80H when a number is received from the ADC (refer to subroutine Analog in Chapter III).

In the case of the "bang-bang" input of the plant (± 10 Volts) the software sends 00H or FFH directly to the DAC and there is no problem with conversions.

The analog to digital converter, AD570, receives the analog output of the plant (CS) and convert it to an eight bit number. As shown in Figure 4.1 the output of the ADC is connected with port B (pins 18 through 25) of the parallel interface M8255. Port C of 8255 (pins 10 and 17) is used to control the sampling process. When the BLANK and CONVERT input (pin 17) of the ADC goes low the conversion is started. Upon completion of the conversion the DATA READY terminal (pin 11) goes low and the data is available at the output. The BLANK and CONVERT input must become high again to prepare the device for the next conversion. Thus, the software has to control these two lines to get the data at the appropriate time.

Table 6 can be used to relate the data inside of the micro and at the ADC output. The two converters must be adjusted to have the same correspondence between

TABLE 6

NUMBERS INSIDE AND OUTSIDE THE MICROPROCESSOR

Analog	ADC	Inside Micro (2's compl.)		
(volts)	Binary	Binary	Hex	Decimal
+4.96	11111111	01111111	7F	+127
+0.39	10001010	00001010	0A	+10
+0.039	10000001	00000001	01	+1
0	10000000	00000000	00	0
-0.039	01111111	11111111	FF	-1
-4.61	00001010	10001010	8A	-118
-5	00000000	10000000	80	-128

the numbers since the whole system must be compatible. In order to guarantee that DAC and ADC are tuned, some simple tests were done. One test consists in applying a DC voltage at the ADC input and send this voltage to the DAC output using a small program. This program is written in machine language and can be loaded into the RAM, using the monitor. An example is presented in Table 7 . The voltage measured at the analog output (V) must be the same as that applied at the analog input (CS), if the resistors in the operational amplifier are equal to 2.5KΩ . This voltage can be varied and observed at the plant input (V).

The small program is executed with the command G15B0.FFFF<ret>. This command guarantees that the last address is included and the program will be in loop. The second address in the command could be anyone greater than 15C9. Some lines of this routine need more explanation:

1. Lines 1 and 2 set up the 8255 to transmit data through ports A and C and receive data in port B. In other words, ports A and C are outputs and port B is input.
2. In lines 3 and 4 a zero is sent to port C (PC7 or pin 17), in order to drive the BLANK CONVERT control of the ADC to low, enabling the conversion.
3. In line 5 the DATA READY pin is checked to verify if the data is already converted. If it is not, the polling process continues in the loop described in lines 5,6 and 7.
4. In line 8 the data is retrieved from port B and in line 9 they are sent to port A, where the DAC is connected.
5. In line 11, port C receives 80H. That means, pin PC7 receives 0, disabling the conversion and preparing the ADC for the next one.

TABLE 7

COMPATIBILITY TEST BETWEEN ADC AND DAC

#		Assembly Language	Comments	Address	Mach. Lang.
1		LD A,83	control word	15B0	3E83
2		OUT (03H),A	8255<--c. word	15B2	D303
3	LOOP:	LD A,00H	A<-- 00H	15B4	3E00
4		OUT (02H),A	start conversion	15B6	D302
5	WAIT:	IN A,(02H)	A<--DATA READY	15B8	DB02
6		CP 0	IS DATA READY?	15BA	FE00
7		JP NZ,WAIT	if not,try again	15BC	C2B815
8		IN A,(01H)	A<-- data	15BF	DB01
9		OUT (00H),A	DAC<-- data	15C1	D300
10		LD A,80H	A<-- 80H	15C3	3E80
11		OUT (02H),A	disable ADC	15C5	D302
12		JP LOOP	repeat process	15C7	C3B415

E. THE PLANT

The plant is represented by an analog simulator as depicted in Figure 4.7 . The input of the plant is a "bang bang" control voltage. The output is a voltage that represents the robot arm position, CS. The input V comes from the DAC as discussed in the last section and can be $+V_{sat}$ or $-V_{sat}$ ($\pm$ 10 Volts in our case).

The design of the analog plant is straight forward and the method is found in most of the classical control books. The approach chosen is based in [Ref. 3]. As shown in Figure 4.6 the plant transfer function can separated in two blocks.

Mathematically we have:

$$\frac{X(s)}{U(s)} = \frac{1}{s+20.55} \rightarrow X(s)(s+20.55) = U(s)$$

$$sX(s)+20.55X(s) = U(s) \rightarrow \dot{x}+20.55x = u.$$

Then, $\dot{x} = u - 20.55x$ and $y = 273.3\int xdx$

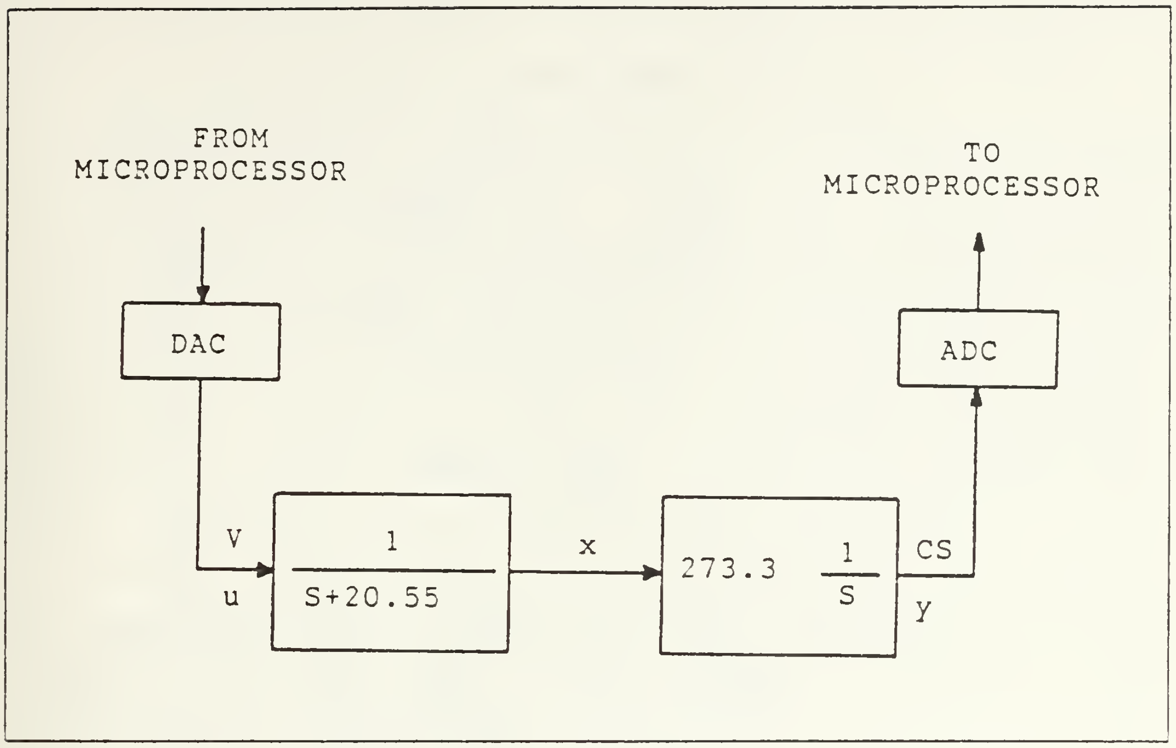

Figure 4.6 Plant Transfer Function.

The above equations provide all the information needed to implement the hardware. If $\dot{x}$ is assumed to be available (refer to Figure 4.7), the next step is to design an integrator to obtain x. Using a capacitor of 1μF and a resistor of 1MΩ the output of the integrator will be $-x$. The 10 KΩ potenciometers that appear in all operational amplifiers are used for off-set adjustments.

Since $-\dot{x}$ and V are available, a summer with a gain of 20.55 for $-x$ and 1 for V yields $-\dot{x}$ at the output. An inverter changes the sign of $-\dot{x}$ and the result is the $\dot{x}$ needed for the feedback to the starting point. The output y or CS is obtained from $-x$ by integrating this variable with gain 273.3.

The design of the plant was quite simple, but the implementation needed special attention. If the components are just put together without any care the result is catastrophic. Oscillations and drifts are the common problems. In the particular case of the integrators the critical points are the capacitor leakage and the input offset error. The integral of the DC offset voltage appears at the output like a ramp voltage, as explained in [Ref. 6]. The power supply can be a source of noise and each pin that is connected to a positive or negative voltage must have a capacitor to the ground. The

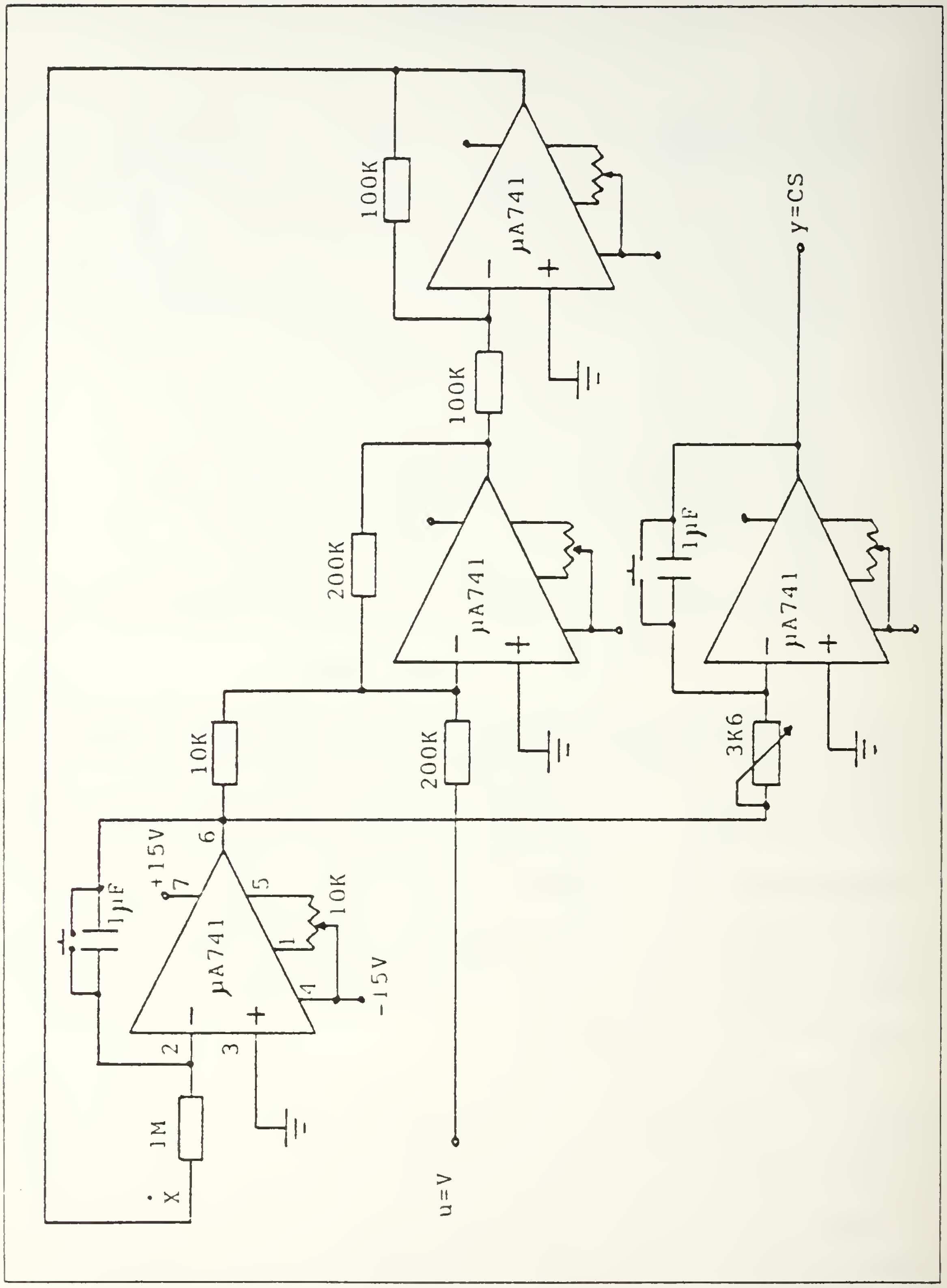

Figure 4.7 Analog Plant.

material the capacitors are made of is a very important topic as pointed out by Roberge in [Ref. 4]. According to the specific application the capacitors must be:

1. Teflon or polystirene for the feedback capacitors (integrators)
2. Solid tantalum electrolytic (greater than $1\mu F$) for the positive and negative terminals of the power supply
3. Mica or glass (0.01 or $0.1\mu F$) for the power connections of each individual chip.

The offset adjustments were done by connecting the inputs of the particular chip to the ground and adjusting the 10 KΩ potentiometer to obtain zero volts at the output. The inputs were grounded before the input resistors to guarantee that the drift voltage produced at the input due to these resistors were cancelled by the appropriate adjustment. In the case of the integrators, the capacitors must be discharged before the adjust. The switches in parallel with these capacitors (shown in Figure 4.7) are used for this purpose. They are also used to reset the integrators just before running the system. In this case, an analog switch, automatically commanded by the software should be desired and was designed. The recommended switch is the LF 11332. The supply could not provide this component.

The plant was submitted to some tests to make sure the transfer function is being implemented. One of the tests consists in building a double integrator with an overall gain of 100. This particular implementation tests the integrator blocks in terms of drifting, oscillations and the integral operation itself. Each integrator has a gain of 10 as shown in Figure 4.8 .

The first integrator is driven by the square wave v(t) with a frequency of 100 Hz. The integration of the square wave results in an inverse triangular wave due to the minus sign of the integrator. Computing the output value of the first integrator at time $t = 5$ msec:

$$x(t) = -\frac{1}{RC}\int_0^T 5dt = -10\int_0^{0.005} 5dt = -0.25$$

$$x(t) = -at, \text{ where } a = 50 \text{ V/sec or } 0.05 \text{ V/msec}$$

$$y(t) = -\frac{1}{RC}\int_0^T -atdt = 10\frac{aT^2}{2} = 0.625 \text{ mV}$$

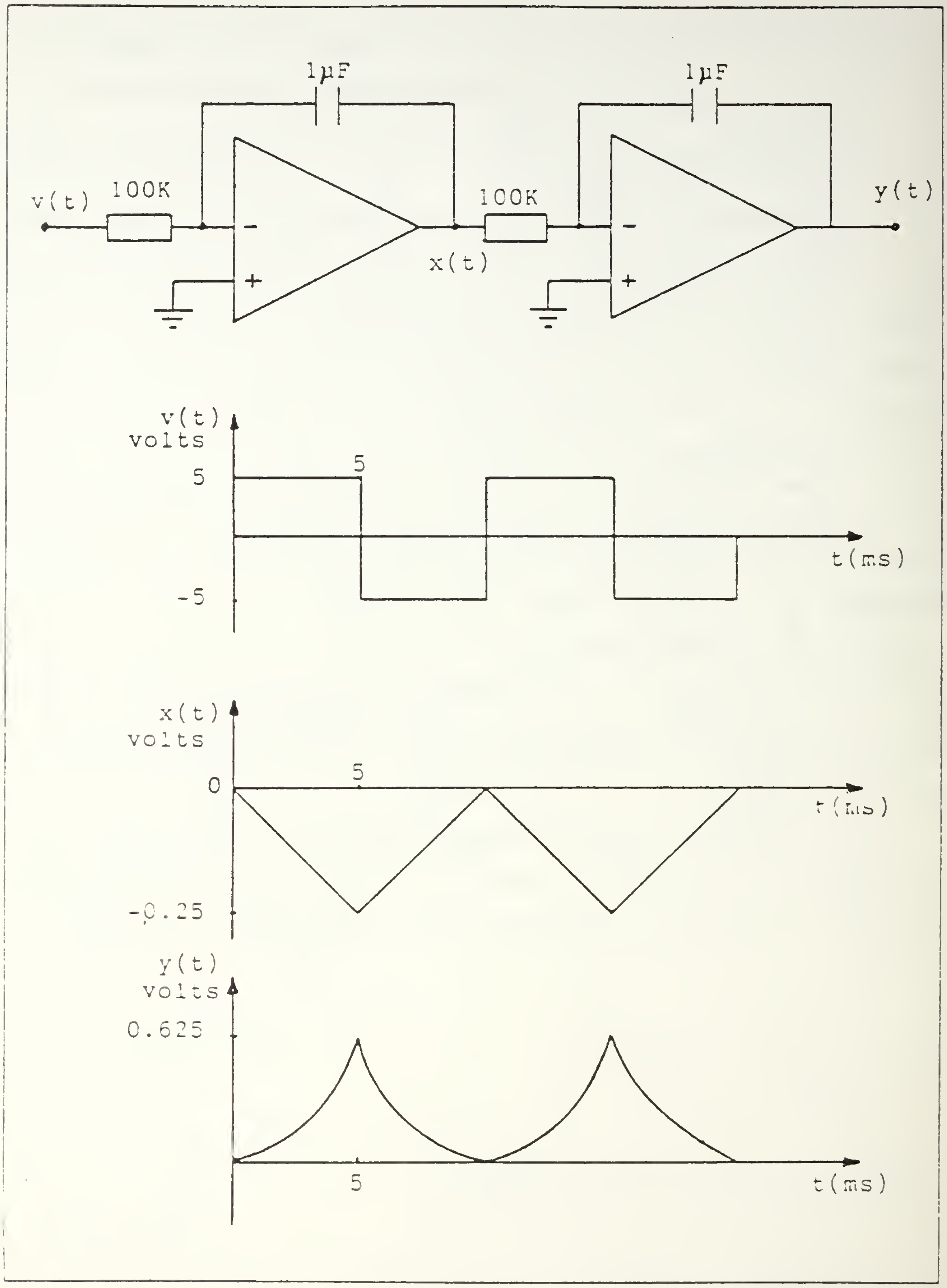

Figure 4.8 Double Integrator.

Once the double integrator is working well, a good test can be done to compare the performance of the digital model and the analog double integrator, since the model is also a double integrator with a gain (Km). So, if the gain Km is set to 100, both double integrators are supposed to have the same behavior. This is a very interesting test because the whole digital servo and interfaces are also checked. The test was carried out and the results displayed on the screen. In despite of the manual reset of the integrators the test was a success, that is, the values of CS (analog double integrator output) and CM (digital model output) were pretty close during all the time. The set up for this test is is shown in Figure 4.9 . The differences between this implementation and the whole system is that in this scheme the plant is just a double integrator and the model parameters are not being updated.

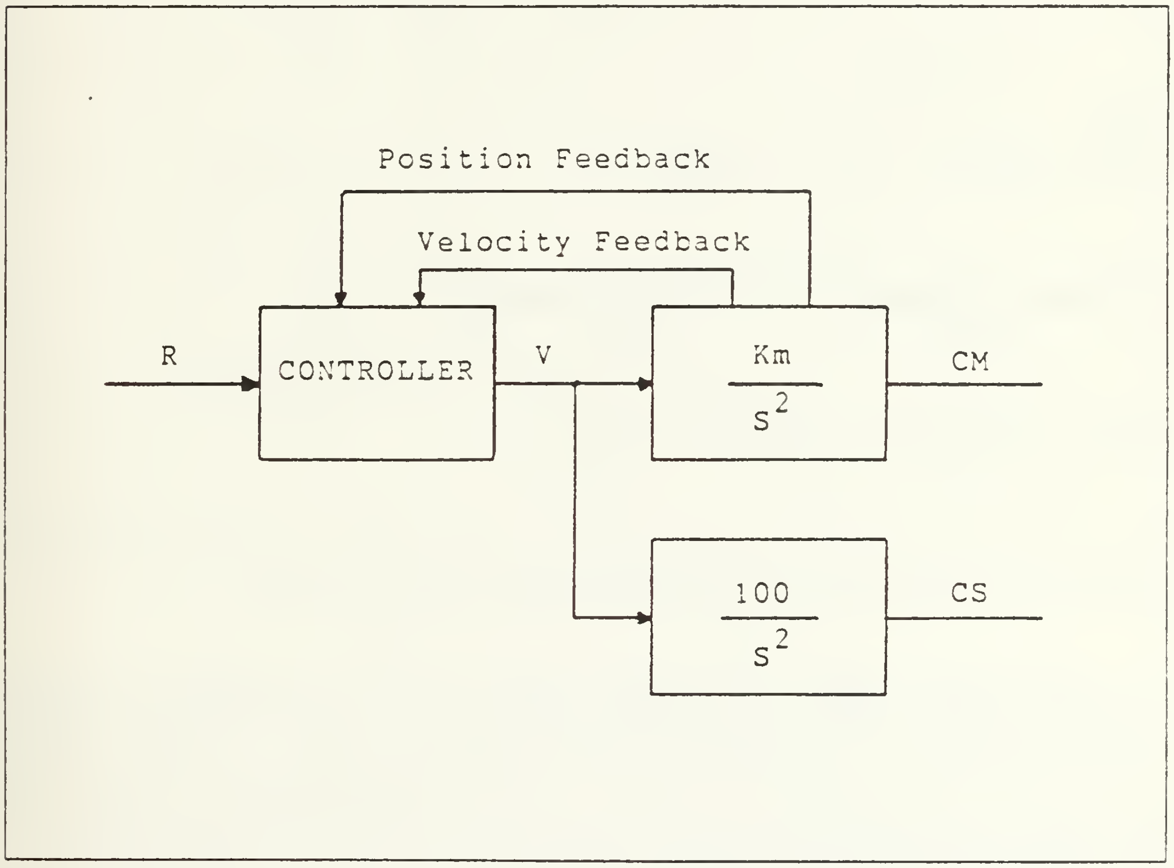

Figure 4.9 Comparison between Digital and Analog Integrators.

The entire analog plant can be tested and the results compared with the theory. Refering to Figure 4.6 one can see that the plant can be separated in two blocks. The first block has input u and output x and the second block has input x and output y. These points are directly available in the circuit (refer to Figure 4.7).

The second block can be tested in the same way used to test the double integrator. Applying a square wave at input x we obtain a triangular wave at output y. The theoretical results are presented in Figure 4.10 and can be obtained as follows:

$$y(t) = -273.3 \int_0^T x(t)dt = -273.3 \int_0^{0.005} 5dt = -6.83 \text{ V}$$

The test of the first block is not so trivial. The output is an exponential function for a step input as shown in Figure 4.11 . The Laplace transfer function of the block is:

$$\frac{X(s)}{U(s)} = \frac{1}{s+20.55} \rightarrow X(s) = \frac{1}{s+20.55} U(s)$$

Applying a step input of 10 V, U(s)= 10/s, we obtain:

$$X(s) = \frac{10}{s(s+20.55)} = \frac{10}{20.55}\left(\frac{1}{s} - \frac{1}{s+20.55}\right)$$

Then, $x(t) = (10/20.55)[1(t) - \exp(-20.55t)] = 0.49[1 - \exp(-20.55t)]$

The theoretical results are presented in Figure 4.11 . The input frequency was set to 1Hz to permit the total excursion of the exponential wave. It is important to sincronize the oscilloscope with the output to obtain a stable image on the screen. The results found in the practical experiment matched very well with those encountered in the theory. In all the tests the integrators are supposed to be reset right before the test.

The gain of 273.3 can be splited between the last integrator and the adder to improve the performance of the analog computer. In this case the circuit presented in

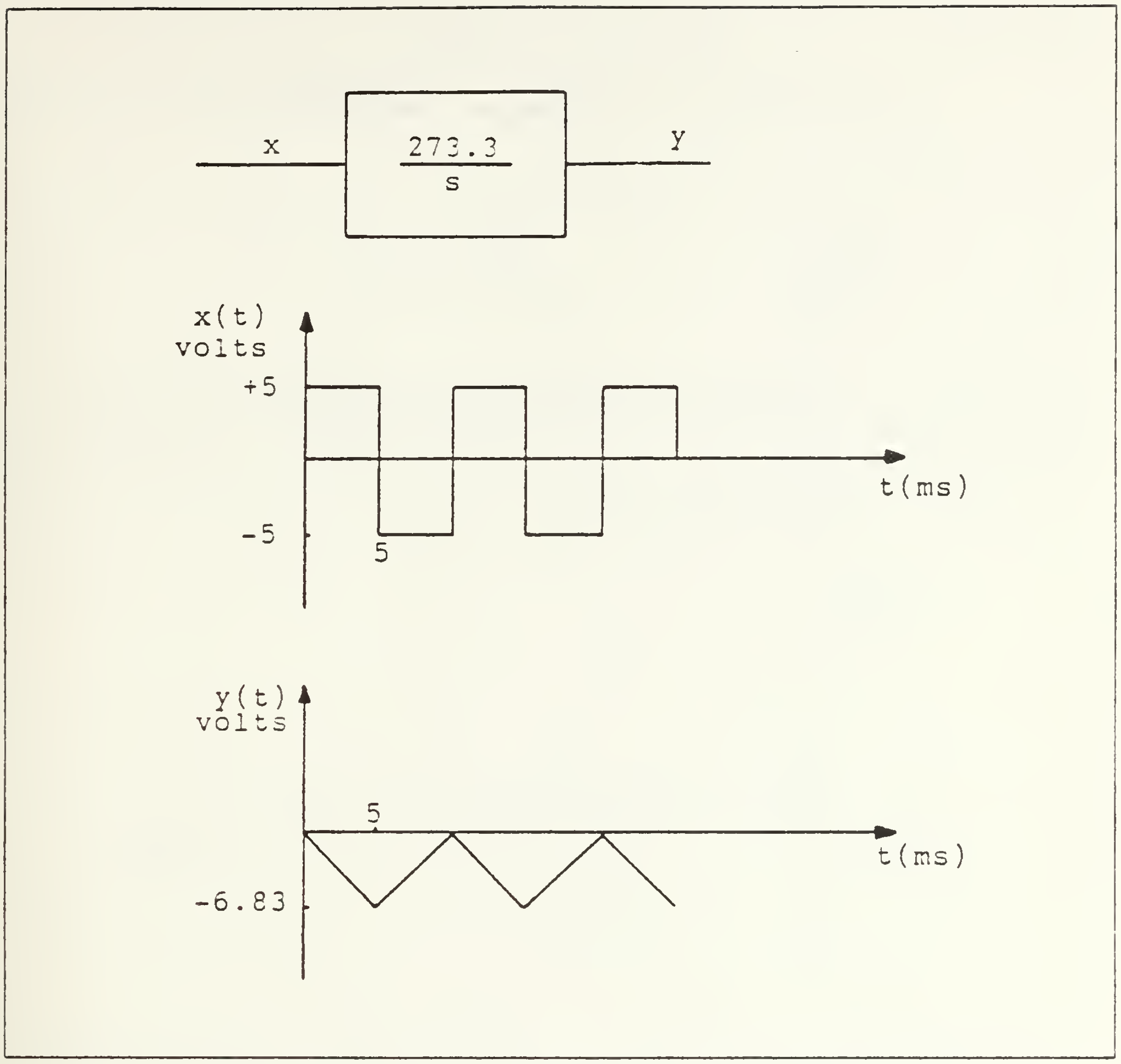

Figure 4.10 The Second Integrator.

Figure 4.7 would have the 3K6 resistor replaced by a 36KΩ resistor in the last integrator (the gain will be 27.3) and the 200KΩ resistor at the adder replaced by a 20KΩ resistor (the gain will be 10).

Another good and easy test that can be done to verify the plant design is to apply a step input, say 1 Volt, and measure the output with a strip recorder. The equivalent plot can be done in the mainframe using the programs "Controls" or "Ewald" and the results can be compared. This test was carried out and the plots turned out to be very similar.

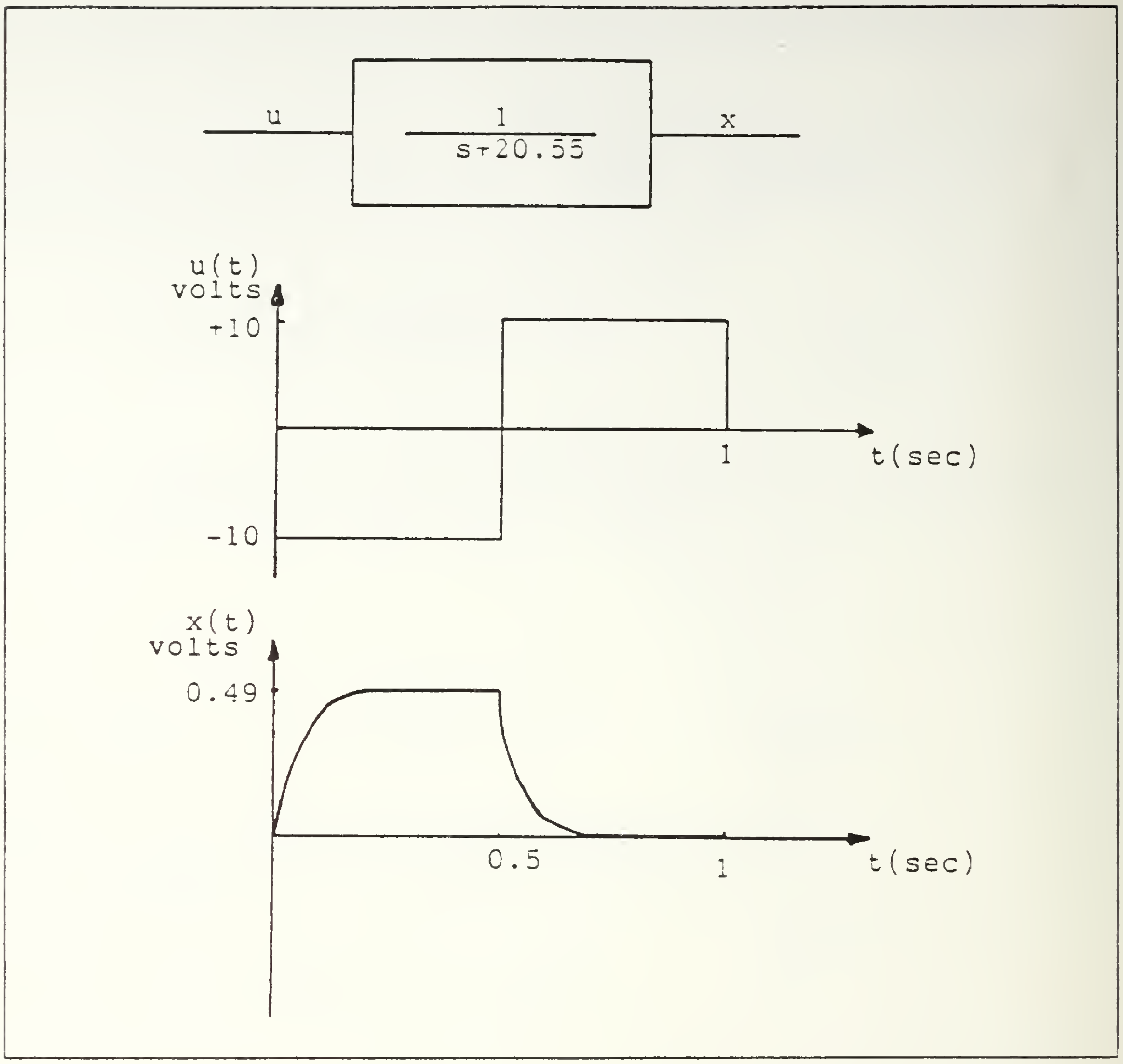

Figure 4.11 Testing the First Block of the Plant.

The analog plant just discussed represents a motor of a disk driver. A second transfer function, representing a robot motor, was also implemented and tested. This transfer function was studied in [Ref. 7] and can be written as

$$G(s) = \frac{9.88}{s(\frac{s}{9100} + 1)(\frac{s}{0.019} + 1)}$$

This transfer function has a real pole at -9100 that can be neglected and another real pole very close to the origin (-0.019) that can be approximated to zero. Therefore, for practical applications, the transfer function can be written as

$$G(s) = \frac{10}{s^2}$$

This plant was implemented in hardware by using two operational amplifiers in a very similar way as that showed in Figure 4.8 . The only difference is that in the second integrator the resistor is 1 $M\Omega$, rather than 100 $K\Omega$. The gain of 10 is obtained in the first integrator to allow a direct test point for the velocity at the output of this stage.

V. PERFORMANCE OF THE SYSTEM

A. THE SCALING PROBLEM

As discussed in Chapter II, the input of the system is a commanded step that determines the position to be achieved by a robot arm or a disk driver arm. The actual position of the arm (CS) is represented in this research by the voltage at the analog plant output, which simulates the transfer function of the motor and load. So, this voltage is a parameter to be measured and converted to an angle in order to compare with the desired position and determine the performance of the system.

Another requirement of the system is the curve following process, that is, the acceleration of the arm must be maximum until the velocity reaches the deceleration curve of the motor and from this point it must follow the curve. The velocity can be obtained from the analog plant by connecting an inverter at the input of the last integrator.

At this point, it becomes necessary to explain the scaling problem between the analog and the digital world involved in this research. The plant input dimension is volts and the output dimension is radians. Therefore, one volt at the plant output represents an angle of one radian. Since the analog to digital converter is an eight bit interface driven by a 5 Volts source, 39 millivolts in the plant output is converted to 1 at the digital output of the ADC. This conversion and other examples are illustrated in Table 8 .

Based on this table the input R applied to the system is a multiple of 2.23 degrees. Also, the number that represents the gain constant, Km, in the program must take the scaling factor into account. Thus, for a Km equal to 300 radians per second we will have, in decimal representation:

$$Km = 300 \times 57.2 / 2.23 = 7708$$

With the actual ADC the system can handle angles from 2.23 degrees to 284.2 degrees. This resolution can be improved by increasing the number of bits in the interface. For instance, if a 12 bit interface is used and the reference voltage is kept the same (5 V), the minimum angle will be 0.14 degrees and the maximum will be 286.34 degrees.

TABLE 8

THE SCALE PROBLEM

ADC Input			ADC Output	
CS(Volts)	CS(radians)	CS(degrees)	Decimal	Hexadec.
0.039	0.039	2.23	1	1
0.31	0.31	17.9	8	8
0.62	0.62	35.8	16	10
1.25	1.25	71.6	32	20
2.50	2.50	142.7	64	40
3.11	3.11	179.0	80	50
4.38	4.38	250.7	112	70
4.96	4.96	284.2	127	7F

B. RESULTS

As pointed out earlier, the system receives an input command from the keyboard and the analog output of the plant must respond as quickly as possible by using a curve following scheme for the velocity. In the case of this research, the performance of the system was checked by several means.

During the development phase, while the system was not working as in its final version, the tests were done by using the routine Display, off-line (refer to Appendix C). After a run, all the important variables were presented on the screen and we could analyze what happened in every loop of the program. Also, the output of the analog plant was observed in the oscilloscope just to verify if the final position was reached or not. In this phase the velocity of the model was considered to be the same as the velocity of the plant. They are supposed to be similar if the algorithm works well.

After the system started operating well under the verification tests mentioned above, a strip recorder was used to check the position and velocity of the plant. The position is readily available from the analog plant output and the velocity is obtained by taking the input of the last integrator (− CSDOT/27.3) and driving it through a amplifier (and inverter) to get the actual velocity.

The voltages obtained in both cases , position and velocity, are then converted to radians and radians per second, respectively. The tests were carried out for both plants (disk driver motor and robot motor), using several inputs (R). The results are summarized in Figures 5.1 through 5.10 . In all plots the velocity of the strip recorder was fixed in 125 mm sec (maximum available). The plots are presented in the same scale they were obtained from the plotter.

Looking at Figure 5.1 we can see that there is some overshoot in the position (CS) plot. In a real application the bang-bang control would be replaced by a linear compensator when the position is reached and this would prevent the overshoot or would reduce it to an acceptable value. In all the velocity plots it can be noticed that the full acceleration process occurs approximately over half of the trajectory. At the maximum point of the curve the actual velocity of the model (CDOT) or plant (CSDOT) crosses the curve of the desired velocity (XDOT). From this point, the deceleration curve of the motor is followed and the velocity drops following XDOT.

In some cases the actual velocity (CSDOT) stays a little bit greater than XDOT and no "chattering" is observed as can be seen in Figures 5.1 and 5.2, for instance. In other cases CSDOT alternates being greater or less than XDOT and the "chattering" can be noticed as in Figures 5.4 and 5.5 , for instance.

The commanded input in the keyboard is an hexadecimal quantity (last column of Table 8) and using the conversion presented in the Table we can label the plots in radians and radians per second. It is easily seen from the plots that the robot plant is much slower than the disk driver plant, as can be confirmed by comparing the abcissas (time) in the position plots and the ordinates in the velocity plots.

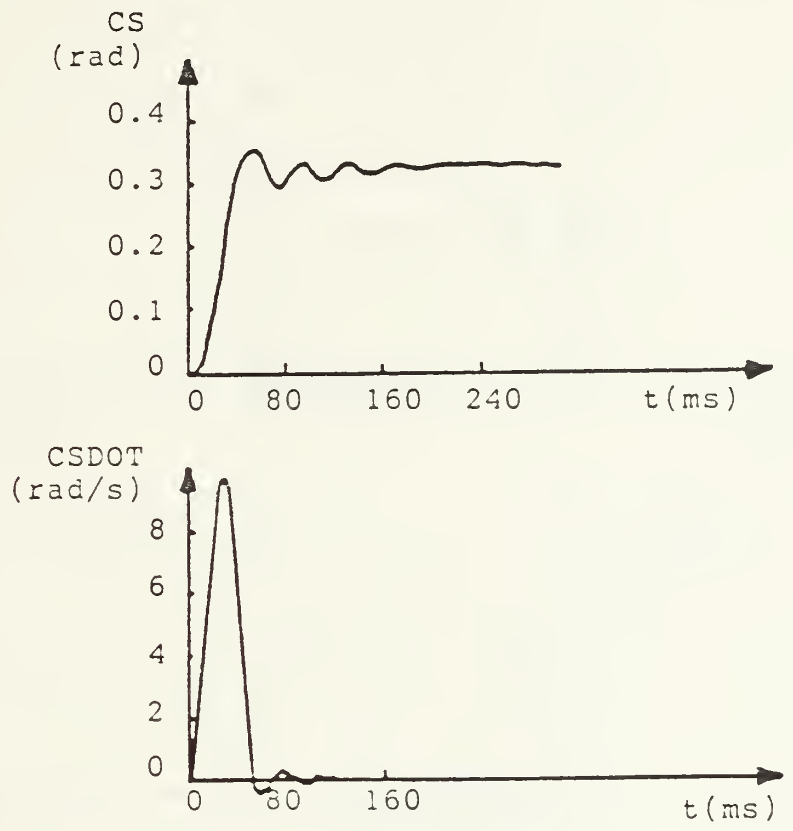

Figure 5.1 Position and Velocity for a Disk Driver Motor, R = 0.31 rad.

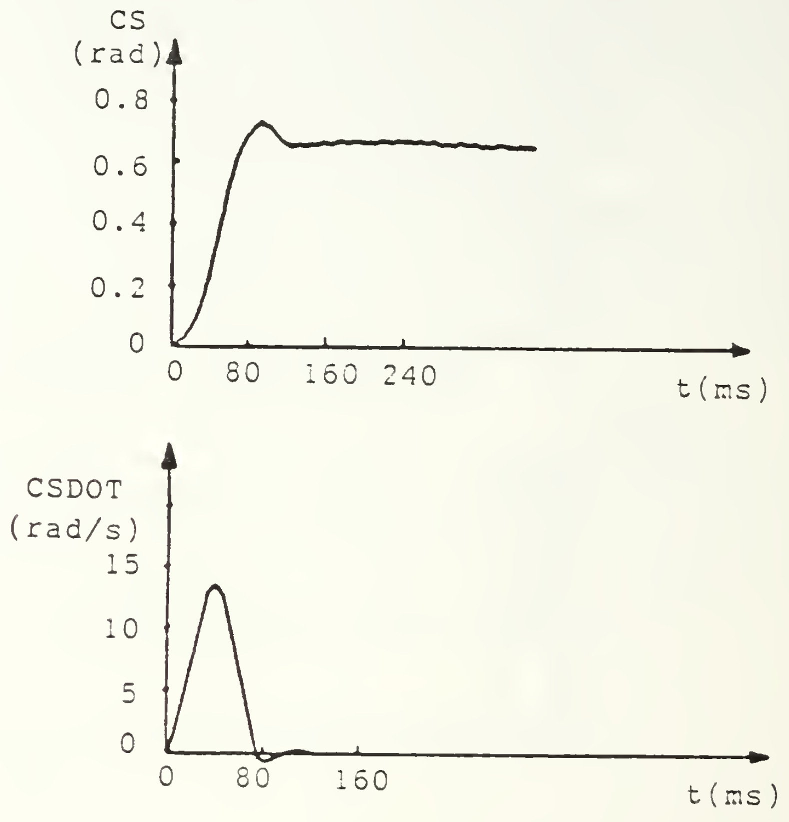

Figure 5.2 Position and Velocity for a Disk Driver Motor, R = 0.62 rad.

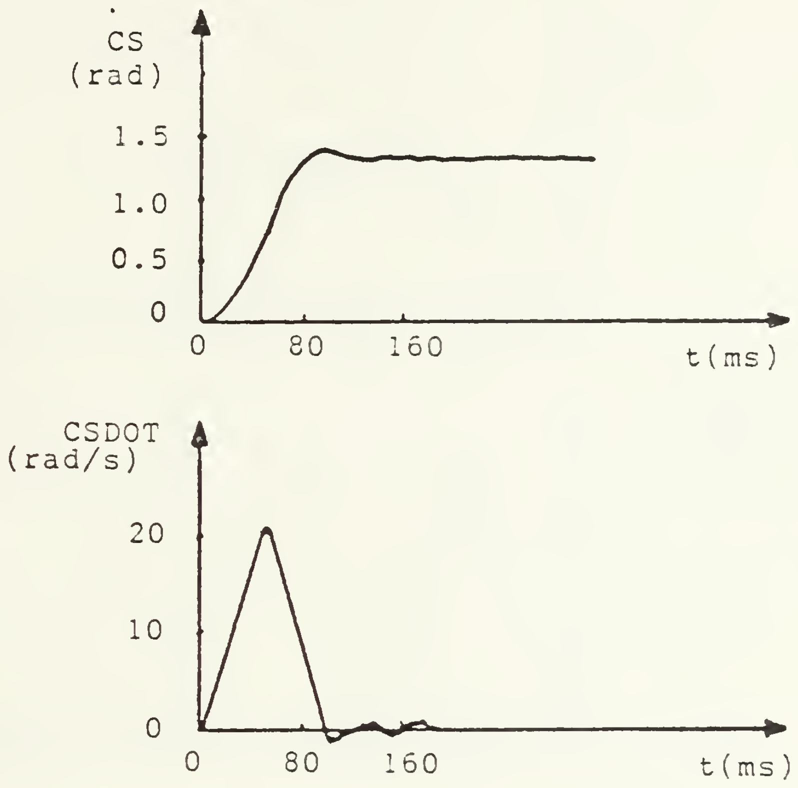

Figure 5.3 Position and Velocity for a Disk Driver Motor, R = 1.25 rad.

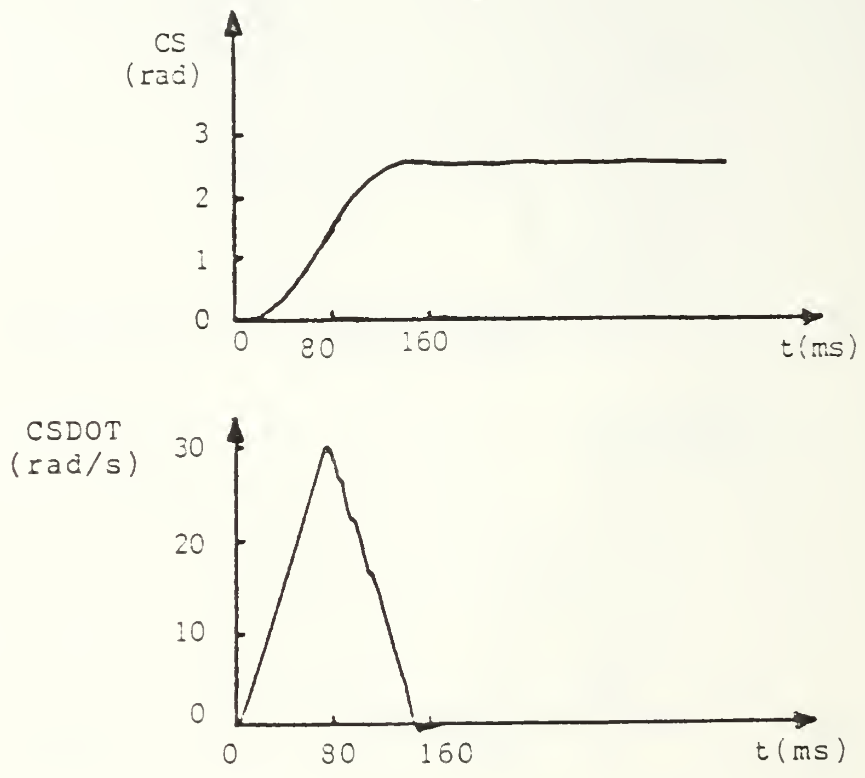

Figure 5.4 Position and Velocity for a Disk Driver Motor, R = 2.50 rad.

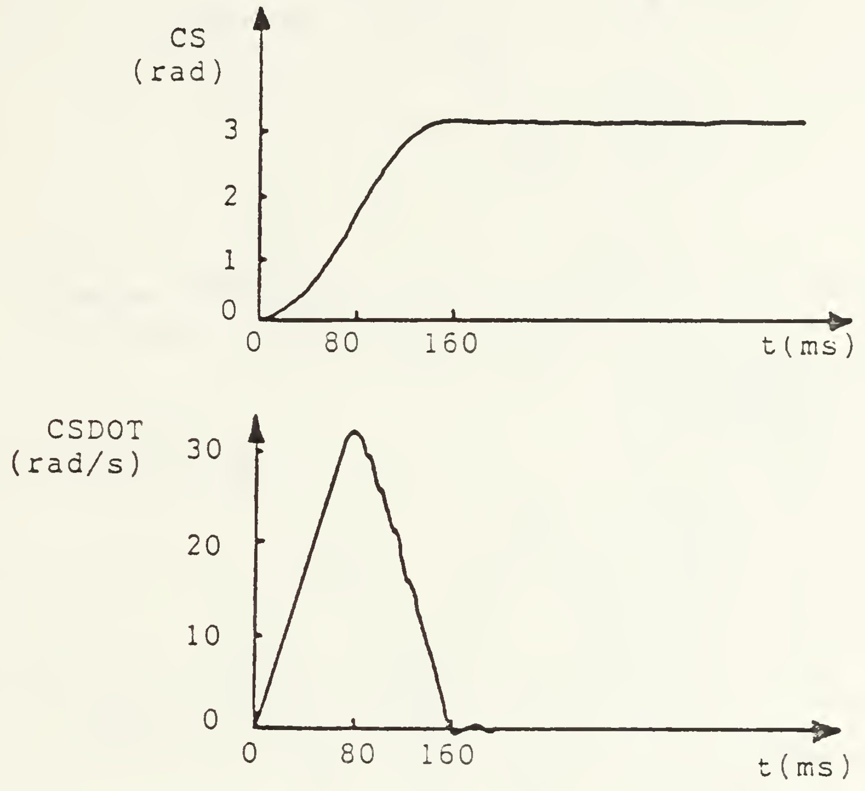

Figure 5.5 Position and Velocity for a Disk Driver Motor, R = 3.11 rad.

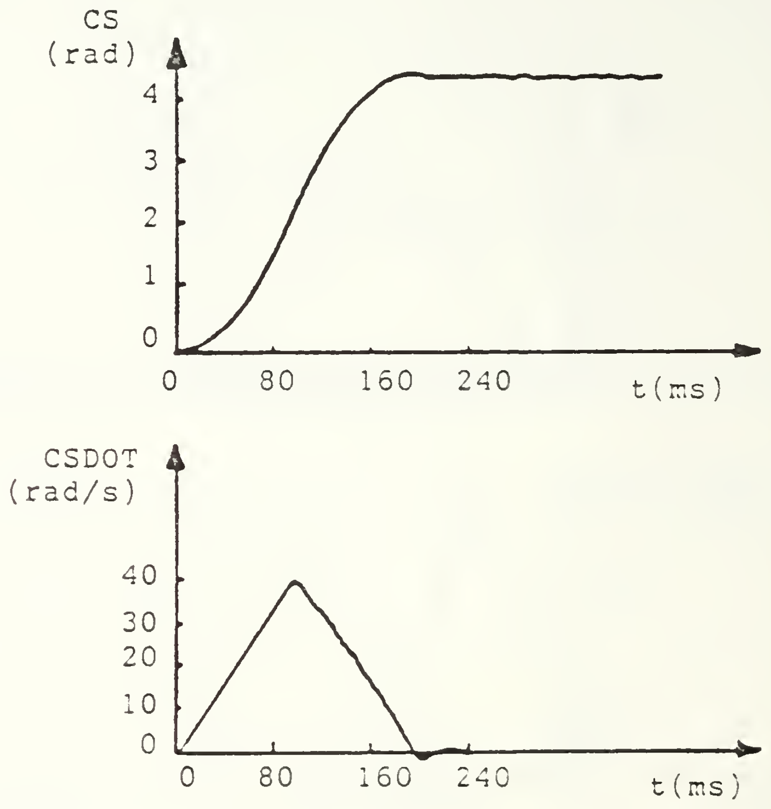

Figure 5.6 Position and Velocity for a Disk Driver Motor, R = 4.38 rad.

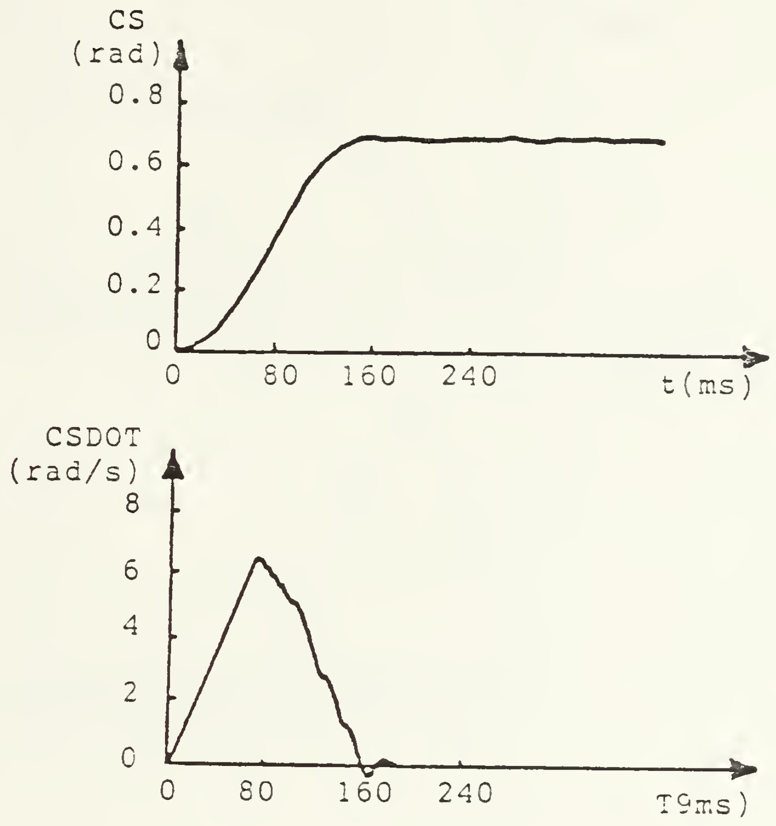

Figure 5.7 Position and Velocity for a Robot Motor, R = 0.62 rad.

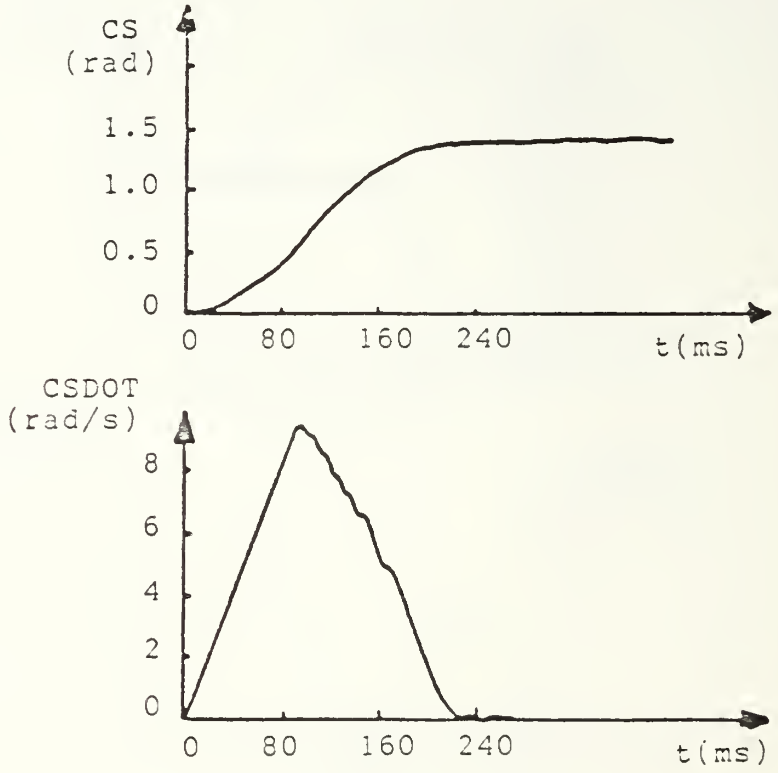

Figure 5.8 Position and Velocity for a Robot Motor, R = 1.25 rad.

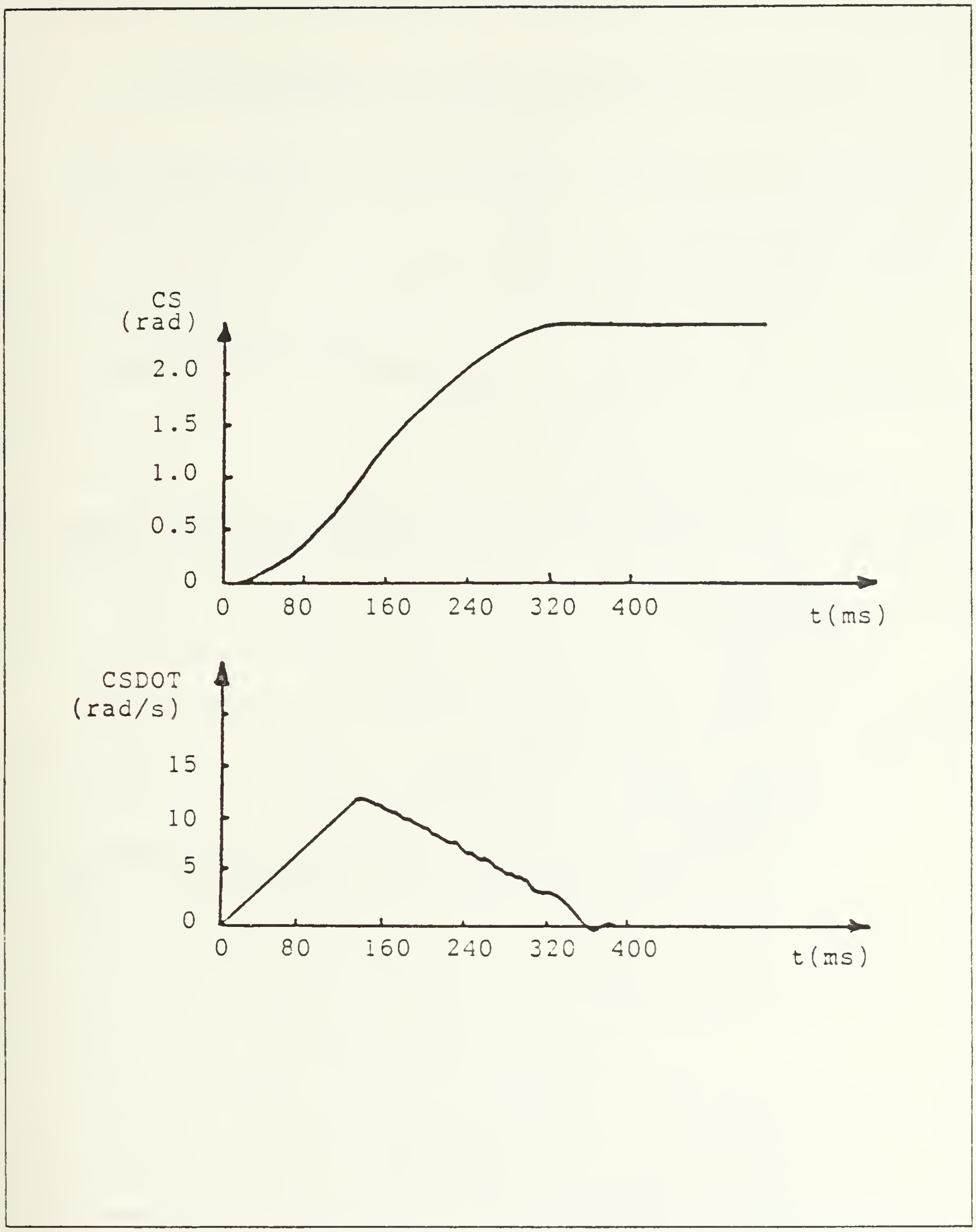

Figure 5.9 Position and Velocity for a Robot Motor, R = 2.50 rad.

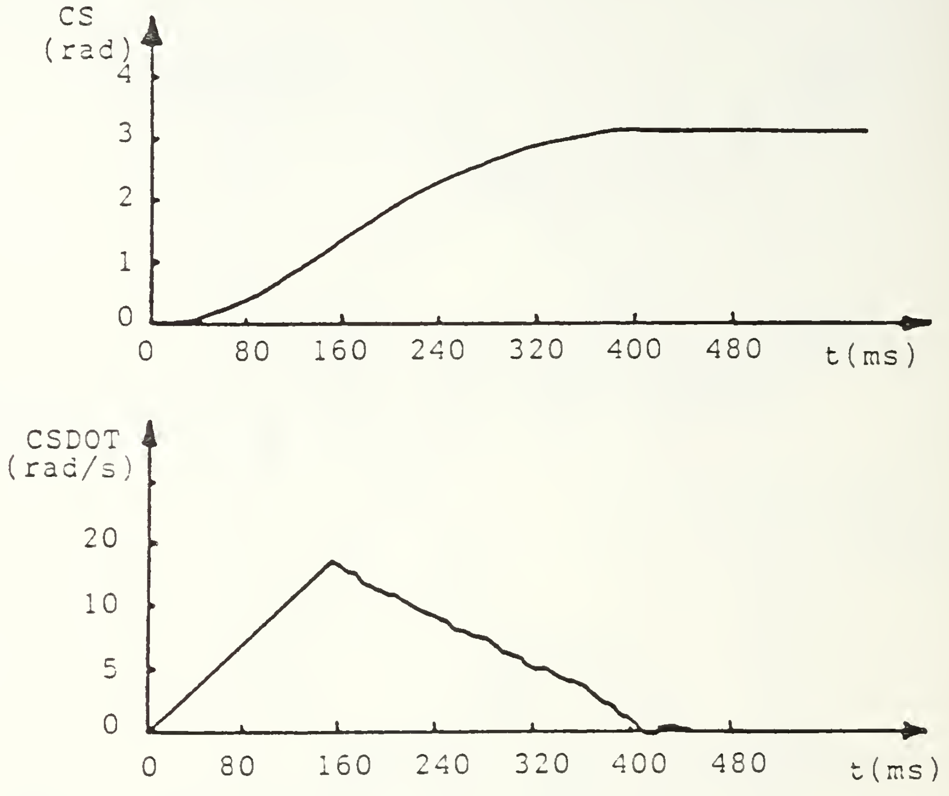

Figure 5.10 Position and Velocity for a Robot Motor, R = 3.11 rad.

VI. CONCLUSIONS / AREAS FOR FURTHER STUDIES

The control of a robot arm or disk driver arm in minimum time (curve following) and autoadaptive was implemented using a microprocessor. The software was designed to work as a servo mechanism with curve following and to implement the algorithm that updates the model based on samples coming from the arm position. An analog computer simulated the motor and load of the arm.

Two different transfer functions for the analog plant were tested, one for a disk driver motor and other for a robot motor. The first plant (disk driver) is found in [Ref. 1] and the second comes from [Ref. 7].

The initial goal of this research was to build a model to roughly represent the device (plant) to be actuated and by sampling its output to update the model in order to minimize the error between them (model and device). Once the model is a "copy" of the plant it can be controlled by using velocity feedback, position feedback and curve following in the model and then applying the same input to both plant and device.

The actual algorithm that updates the output of the model and the "gain constant" Km, does not update the velocity. The technical reasons why the velocity is not being updated were explained in Chapter III, but the fact is that the lack of velocity updating did not influence the performance of the system at all since the velocity computed for the model is based upon the updated value of Km and in an indirect way, the velocity is being updated.

The commanded input was applied from the keyboard in order to easily check the system for different inputs. However, as pointed out in Chapter I, this input can come from a central computer and, in this case, several systems like the one described here could be used to actuate different arms.

The results presented in Chapter V show that the system works and can be used in a real application. It was also pointed out that it can be improved. One of the aspects that can be worked out is the interface. The system can go from a resolution of 2.23 degrees (eight bits) to a resolution of 0.14 degrees by just changing the interface (ADC) to a twelve bit ADC. For a sixteen bit ADC the resolution would go to 0.0087 degrees. The problem here is that at this level the noise starts corrupting the results and additional arrangements (filters, isolations, etc.) must be incorporated.

Another improvement in the performance of the system would be to transform the whole program to a 32 bit program. The impact of this change would be in the accuracy of the system. The actual program does several operations using 32 bit floating point features of the Arithmetic Processing Unit as in the curve computation and in the trapezoidal integration. One concern about the use a 32 bit program is the time. However, a good measure to mininize the execution time is to cut off all the subroutine calls and imbed them into the main program.

Another important topic for a future research is to connect the microprocessor with a real motor and arm and increase the resolution to 12 bits. Also, the movement of the arm coud have two stages: a large movement, using a resolution of 2.23 degrees (8 bits) and a fine adjustment using a resolution of 0.14 degrees (12 bits). The 8 bit interface would allow a faster manipulation of the data during most of the trajectory.

The basic idea and the skills necessary to implement a new hardware / software or improve the one reported here were provided in this research. However, the best legacy of this thesis is the proof that, an autoadaptive algorithm applied in a real time programming for disk drivers and robots can be used succesfully.

APPENDIX A
PROGRAM MODEL - 16 BITS

```
 .Z8D
EXTRN           SCRLF,GETSTRIN,HEXCONV,GHEXERR,R,CM,CDOT,CDDOT,CDM1,CDDM1,KM
EXTRN           N,V,CS,CSM1,CSM2,NUMBER,QUOT,ONES,TENS,HUNDREDS,THOUS,TTHOUS
EXTRN           RSSTAT,RECV,MESSAGE,MONITOR,HELLO,BOUT,TRAP3D,MONLOOP,SPACES
EXTRN           ECHO,ASCONV,TEMPIY,CMCF,CF,MFLAG,NN,KMFLAG,NS
PUBLIC          MODEL,TRANSFER,DISPLAY
;
ZERO            EQU     0
T1              EQU     D6BCH           ;INTEGRATION STEP (T1=2/T)
PVSAT           EQU     0AH             ;POSITIVE SATURATION LIMIT=+1D
MVSAT           EQU     DFFF6H          ;NEGATIVE SATURATION LIMIT=-1D
T1OCF           EQU     12H             ;T1/CF=18, T1=180D AND CF=10D
KMC             EQU     381CH           ;KMC=CONSTANT IN THE ALGORITHM
K1A             EQU     8CH             ;0.8*SQRT(2.VSAT.KM)
;
;THIS PROGRAM SIMULATES A SERVO SYSTEM.
;THE VARIABLES HAVE THE FOLLOWING MEANING:
;R ....... STEP INPUT OR POSITION TO BE ACHIEVED BY THE OUTPUT
;CM ...... POSITION OUTPUT
;CDOT .... VELOCITY
;CDDOT ... ACCELERATION
;CDM1 .... PREVIOUS VALUE OF THE VELOCITY
;CDDM1 ... PREVIOUS VALUE OF THE ACCELERATION
;
;
MODEL:          CALL    SCRLF
                LD      A,83H            ;PORT A IS OUTPUT TO D/A,
                OUT     (03H),A          ;PORT B IS INPUT FROM A/D, PORT C
                LD      A,7FH            ;UPPER IS OUTPUT,LOWER IS INPUT
                OUT     (00H),A          ;RESET THE D/A OUTPUT, XDOTE=D
                LD      A,8DH            ;
                OUT     (D2H),A          ;DISABLE A/D CONVERTER
                LD      B,1DH            ;THIS BLOCK RESETS THE VARIABLES
                LD      HL,CM            ;CM,CDOT,CDDOT,CDM1,CDDM1,CSM1,CSM2
LOOP:           LD      (HL),ZERO        ;AND CMCF
                INC     HL
                DJNZ    LOOP
;
                LD      HL,0D64H
                LD      (CF),HL          ;CORRECTOR FACTOR<--- 1D0
                LD      A,0DH
                LD      (MFLAG),A        ;MODEL FLAG IS SET TO ZERO
                LD      (NS),A           ;NS(STORAGE CONTROL)<--- 0
                LD      (KMFLAG),A       ;ALGORITHM FLAG
                LD      HL,D5DDH         ;KM=256*5=128D
                LD      (KM),HL          ;
                LD      HL,DD01H         ;SET COUNTER TO ONE TO BE USED BY
                LD      (NN),HL          ;WIKSTROM ALGORITHM
                LD      HL,D0H           ;HL<--- D0H
                LD      (NUMBER),HL      ;NUMBER<--- D
                LD      (QUOT),HL        ;QUOT<--- D
                CALL    SCRLF            ;LINE FEED
                LD      IX,HELLO         ;SET UP POINTER TO MESSAGE
                CALL    MESSAGE          ;ASK FOR POSITION INPUT
                CALL    SCRLF
                CALL    GETSTRIN         ;GET CHARACTER FROM KEYBOARD
                CALL    HEXCONV          ;CONVERT INTO HEX
                JP      C,GHEXERR        ;IF ERROR IN THE INPUT
                LD      (R),DE           ;SAVE POSITION IN R
                CALL    SCRLF
                LD      IY,16DDH         ;INIT. DISPLAY ROUTINE POINTER
```

```
PLOOP:          LD      DE,(CM)         ;DE<---OUTPUT POSITION
                LD      HL,(R)          ;HL<---INPUT POSITION
                CALL    SUBTRACT        ;POSITION ERROR=HL<---HL-DE
                CALL    STORE           ;SAVE POSITION ERROR
                LD      C,0             ;SET FLAG TO ZERO FOR POS. NUMBERS
                BIT     7,H             ;IF NUMBER IS POSITIVE,
                JR      Z,POSITIVE      ;GO TO LOCATION "POSITIVE"
                LD      C,1             ;IF NUMBER IS NEG., SET FLAG TO 1
                LD      A,L             ;CONVERT IT IN A POSITIVE NUMBER
                NEG                     ;A<--- 0-A (INVERTS THE A SIGN)
                LD      L,A
                LD      A,H
                CPL
                LD      H,A
POSITIVE:       CALL    CURVE           ;HL<---SQRT(ERROR)*K1*SQRT(2KM.VSAT)
                BIT     0,C             ;IF FLAG IS 0,(THE ERROR WAS POS.)
                JR      Z,OK            ;GO TO "OK"
                LD      A,L             ;THE ERROR WAS NEGATIVE! SO, ...
                NEG                     ;CONVERT IT BACK TO NEGATIVE
                LD      L,A
                LD      A,H
                CPL
                LD      H,A
OK:             CALL    STORE           ;SAVE XDOT
                LD      DE,(CDOT)       ;DE<--- CDOT
                CALL    SUBTRACT        ;XDOTE<--- XDOT-CDOT
                CALL    STORE           ;SAVE XDOTE
                BIT     7,H             ;IF XDOTE IS POSITIVE ...
                JR      Z,PLUS          ;GO TO PLUS
                LD      HL,MVSAT        ;HL<--- -10
                LD      A,0DH           ;A<--- D
                OUT     (D0H),A         ;DAC<--- -1D VOLTS
                LD      A,01H           ;SET KMFLAG TO 1 WHEN XDOTE< 0.
                LD      (KMFLAG),A      ;KMFLAG <--- A
                JR      VOLTS           ;GO TO VOLTS
PLUS:           LD      HL,PVSAT        ;DAC<--- +10
                LD      A,0FFH          ;A<--- FF
                OUT     (00H),A         ;DAC<--- -10V
VOLTS:          LD      (V),HL          ;V IS SAVED
                LD      DE,(KM)         ;DE<--- KM
                CALL    MULTIPLY        ;HL<--- CDDOT=KM*V
                CALL    STORE           ;STORE CDDOT (FOR DISPLAY PURPOSES)
                LD      (CDDOT),HL      ;SAVE NEW VALUE OF CDDOT
;
;TRAPEZOIDAL INTEGRATION : INPUT IS ACCELERATION, OUTPUT IS VELOCITY.
;THIS BLOCK DOES CDOT=CDM1+(CDDM1+CDDOT)*T/2 IN 32 BIT FLOATING POINT
;THE INTEGER IS COMPUTED TO BE DISPLAYED OFF LINE.
                LD      DE,(CDDM1)      ;DE<--- CDDM1
                LD      (CDDM1),HL      ;CDDM1<--- CDDOT
                LD      A,L             ;APU<--- CDDOT
                OUT     (D8H),A
                LD      A,H
                OUT     (08H),A
                LD      A,1DH           ;TOS(APU)<--- FLOAT(CDDOT)
                OUT     (09H),A
                LD      A,E             ;APU<--- CDDM1
                OUT     (08H),A
                LD      A,D
                OUT     (08H),A
                LD      A,1DH           ;TOS(APU)<--- FLOAT(CDDM1)
                OUT     (09H),A
                LD      A,1DH           ;TOS<--- CDDOT+CDDM1
                OUT     (D9H),A
                LD      DE,T1           ;DE<--- 2/T
                LD      A,E             ;APU<--- T1
                OUT     (08H),A
                LD      A,D
                OUT     (08H),A
                LD      A,1DH           ;TOS<--- FLOAT (T1)
                OUT       (09H),A            ;
                LD        A,13H              ;TOS<--- (CDDOT+CDDM1)T/2
                OUT       (09H),A
                LD        DE,(QUOT)          ;DE<---LSBYTE OF CDOT (FLOATING)
                LD        HL,(NUMBER)        ;HL<---MSBYTE OF COOT (FLOATING)
                LD        A,E                ;TOS<--- CDOT (FLOATING POINT)
                OUT       (08H),A
                LD        A,D
                OUT       (08H),A
                LD        A,L
                OUT       (08H),A
                LD        A,H
                OUT       (08H),A
                LD        A,10H              ;TOS<---CDOT+(CDDOT+CDDM1)T/2=CDOT
                OUT       (09H),A
                IN        A,(08H)            ;NUMBER,QUOT<--- FLOAT(CDOT)
                LD        H,A
                IN        A,(08H)
                LD        L,A
                IN        A,(08H)
                LD        D,A
                IN        A,(08H)
                LD        E,A
                LD        (NUMBER),HL
                LD        (QUOT),DE
                LD        A,E                ;RETURN CDOT (FLOAT) TO APU
                OUT       (08H),A
                LD        A,D
                OUT       (08H),A
                LD        A,L
                OUT       (08H),A
                LD        A,H
                OUT       (08H),A
                LD        A,1FH              ;TOS<--- INTEGER(CDOT)
                OUT       (09H),A
                IN        A,(08H)            ;HL<--- INTEGER(CDOT)
                LD        H,A
                IN        A,(08H)
                LD        L,A
                LD        (CDOT),HL          ;SAVE CDOT
                CALL      STORE              ;CDOT IS STORED TO BE DISPLAYED
;
;TRAPEZOIDAL INTEGRATION : INPUT IS VELOCITY
;THIS BLOCK DOES: CM=CM1+(CDMI+CDOT)*T/2
;
                LD        DE,(CDM1)          ;DE<--- CDM1
                LD        (CDM1),HL          ;CDM1 IS UPDATED (CDM1<--- CDOT)
                CALL      ADDITION           ;HL<--- CDM1+CDOT
                LD        DE,T1OCF           ;DE<--- T1/CF=18
                CALL      DIVIDE             ;HL<---(CDOT+CDM1)/T1OF=CDOT*CF/T1
                LD        DE,(CMCF)          ;DE<--- CMCF (CM *100)
                CALL      ADDITION           ;HL=CMCF=CMCF+(CDM1+CDOT)*T/2
                LD        (CMCF),HL          ;CMCF IS UPDATED
                LD        DE,(CF)            ;DE<--- CF
                CALL      DIVIDE             ;HL<---CMCF/CF = CM
;               LD        (CM),HL            ;CM IS UPDATED
;               CALL      STORE              ;SAVE CM (MODEL OUTPUT)
                CALL      ANALOG             ;HL<---CS FROM ANALOG PLANT
                CALL      STORE              ;STORE CS FOR DISPLAY PURPOSES
                CALL      WALG               ;KM AND CM ARE UPDATED
                LD        HL,(NN)            ;HL<--- LOOP COUNTER
                CALL      STORE              ;STORE LOOP COUNTER
                INC       HL                 ;INCREMENT LOOP COUNTER
                LD        (NN),HL            ;SAVE LOOP COUNTER
                LD        A,(NS)             ;A<--- NS
                INC       A
                LD        (NS),A             ;NS<--- NS+1
                CP        01H                ;IS NS=1?
                JR        Z,RESET            ;IF YES, GO TO RESET.
                LD          A,0FFH              ;A<--- FF
                LD          (MFLAG),A           ;MFLAG<--- 11111111
                JR          SCREEN              ;GO TO SCREEN
RESET:          LD          A,(MFLAG)           ;A<--- MFLAG
                CPL                             ;COMPLEMENT THE FLAG
                LD          (MFLAG),A           ;MFLAG IS COMPLEMENTED
                LD          A,0                 ;A<--- 0
                LD          (NS),A              ;NS<--- 0
SCREEN:         LD          (N),IY              ;N<--- IY
                LD          A,(N+1)             ;A<--- HIGH BYTE OF IY
                CP          24H                 ;IS IT 24? (MEMORY ENDS AT 23FFH)
                JR          NZ,CONTINUE         ;IF IT IS NOT, GO TO CONTINUE
                LD          A,55H               ;SET MFLAG TO AVOID FURTHER STORAGES
                LD          (MFLAG),A           ;SINCE THE MEMORIES ARE FULL.
CONTINUE:       JP          PLOOP               ;ENDLESS LOOP
;
;*****************************************************************************
;****************** END OF MAIN PROGRAM **************************************
;*****************************************************************************
;
;THIS SUBROUTINE INPUTS THE ANALOG OUTPUT FROM THE PLANT (CS)
;
ANALOG:         LD          A,0                 ;ENABLE THE ADC CONVERTER
                OUT         (02H),A             ;PORTC<--- 00
HOLD:           IN          A,(02H)             ;A<--- READY LINE FROM A/D VIA P.C
                CP          0                   ;IS A/D READY?
                JR          NZ,HOLD             ;IF NOT, VERIFY AGAIN.
                IN          A,(01H)             ;A<--- CS FROM PORT B
                ADD         A,80H               ;CS IS CONVERTED TO 2'S COMPLEMENT
                BIT         7,A                 ;CHECK OVERFLOW DUE TO NEGATIVE
                JR          Z,GOOD              ;VOLTAGE FROM A/D. IF THERE IS
                LD          A,0                 ;OVERFLOW, A<---0
GOOD:           LD          L,A                 ;L<--- CS
                LD          H,0                 ;H<---0
                LD          (CS),HL             ;SAVE CS
                LD          A,80H               ;A<--- CONTROL WORD TO DISABLE ADC
                OUT         (02H),A             ;PORT C<--- 80H
                RET
;
;THIS SUBROUTINE IMPLEMENTS WIKSTRON ALGORITHM TO COMPUTE KM AND CDOT USING
;THE ANALOG OUTPUT FROM THE PLANT(CS). THEN,CM,CDOT AND KM ARE UPDATED
;KM=2CS/VSAT(NT)**2 AND CDOT=(CS-CSM1)/T. OR, KM=(16529/N**2)*CS*10, WHERE
;T=1.1 MS AND VSAT=10
;
WALG:           LD          A,(KMFLAG)          ;VERIFY FLAG. IF IT IS 1 DO NOT
                CP          0                   ;COMPUTE KM ANYMORE. KEEP THE
                JR          NZ,KMFIX            ;LAST VALUE.
                LD          HL,KMC              ;HL<--- KMC=2/VSAT*T**2*10
                LD          DE,(NN)             ;DE<--- COUNTER
                CALL        DIVIDE              ;HL<--- KMC/N
                LD          DE,(CS)             ;DE<--- CS
                LD          A,E                 ;A<--- CS
                CP          0                   ;
                JR          Z,KMFIX             ;IF CS=0, KEEP INITIAL KM
                CALL        MULTIPLY            ;HL<---(KMC/N)*CS
                LD          DE,(NN)             ;DE<--- NN
                CALL        DIVIDE              ;HL<--- (KMC/N**2)*CS
                LD          DE,0AH              ;DE<--- 10
                CALL        MULTIPLY            ;HL<--- (KMC/N**2)*CS*10=KM
                LD          A,H                 ;A<--- MSBYTE OF KM
                CP          0CH                 ;IF KM > 3072, SET KM TO 0CCCH
                JP          M,KMSMALL           ;ELSE, GO TO KMSMALL
                LD          DE,0CCCH            ;DE<--- 0CCCH = 3276
                LD          (KM),DE             ;KM<--- 3276
                JR          KMFIX               ;GO TO KMFIX (DO NOT GO TO KMSMALL)
KMSMALL:        LD          (KM),HL             ;KM IS UPDATED
KMFIX:          LD          HL,(CS)             ;HL<--- CS
                LD          (CM),HL             ;CM<--- CS, CM IS UPDATED
                CALL        STORE               ;STORE NEW VALUE OF CM
                RET
;
;ADDITION ROUTINE USING THE INTEL APU 8231 : HL<--- HL+DE
;
ADDITION:       CALL      OUTOP               ; SEND OPERANDS TO 8231
                LD        A,6CH               ;ADD COMMAND
                OUT       (09H),A
                CALL      INOP                ; GET RESULT AND STORE IN HL
                RET
;
;SUBTRACTION ROUTINE USING THE APU 8231: HL<--- HL-DE
;
SUBTRACT:       CALL      OUTOP               ;SEND OPERANDS TO 8231 STACK
                LD        A,6DH               ;SEND COMMAND SUBTRACT TO 8231
                OUT       (09H),A
                CALL      INOP                ;GET THE RESULT FROM 8231
                RET
;
;MULTIPLICATION ROUTINE USING THE APU 8231: HL<--- HL*DE
;
MULTIPLY:       CALL      OUTOP               ;SEND OPERANDS TO 8231
                LD        A,6EH               ;SEND COMMAND MULTIPLY TO 8231
                OUT       (09H),A
                CALL      INOP                ;GET THE RESULT AND STORE IN HL
                RET
;
;THIS SUBROUTINE COMPUTES (SQRT(ERROR))*K1*SQRT(2.VSAT.KM) USING 32 BITS
;FLOATING POINT OPERATIONS OF THE ARITHMETIC PROCESSING UNIT (APU).
CURVE:          LD        A,L                 ;SEND DATA TO 8231 (16 BITS)
                OUT       (08H),A
                LD        A,H
                OUT       (08H),A
                LD        A,1DH               ;SEND COMMAND TO CONVERT 16 BITS
                OUT       (09H),A             ;INTEGER TO 32 BITS FLOATING POINT
                LD        A,01H               ;SEND SQRT COMMAND
                OUT       (09H),A             ;
                LD        DE,K1A              ;DE<--- K1A=K1.SQRT(2.VSAT.KM)
                LD        A,E                 ;APU<--- K1A
                OUT       (08H),A             ;
                LD        A,D                 ;
                OUT       (08H),A             ;
                LD        A,1DH               ;TOS(APU)<--- FLOAT(K1A)
                OUT       (09H),A             ;
                LD        A,12H               ;MULTIPLY K1A*SQRT(E)
                OUT       (09H),A             ;
                LD        A,1FH               ;CONVERT THE RESULT TO 16 BITS
                OUT       (09H),A             ;
                IN        A,(08H)             ;STORE THE RESULT IN HL
                LD        H,A
                IN        A,(08H)
                LD        L,A
                RET
;
;THIS ROUTINE STORES INTERMEDIATE RESULTS TO BE DISPLAYED OFF LINE
;
STORE:          LD        A,(MFLAG)           ;VERIFY MFLAG STATUS.
                CP        0                   ;IF IT IS 0, STORE DATA.
                JR        NZ,NOSTOR           ;ELSE, DO NOT STORE.
                LD        (IY),L              ;MEMORY<--- DATA (LOW)
                INC       IY                  ;IY IS INCREMENTED
                LD        (IY),H              ;MEMORY<--- DATA (HIGH)
                INC       IY                  ;IY IS INCREMENTED
NOSTOR:         RET
;
;DIVISION ROUTINE USING THE APU 8231: HL<--- HL/DE
;
DIVIDE:         CALL      OUTOP           ;OUTPUT OPERANDS TO 8231 STACK
                LD        A,6FH           ;EXECUTION COMMAND FOR DIVISION
                OUT       (09H),A         ;IS SENT TO 8231
```

```
             CALL      INOP           ;INPUT THE RESULT FROM 8231
             RET
;
;THIS ROUTINE OUTPUTS OPERANDS TO 8231 STACK: NOS<---HL, TOS<---DE
;
CUTOP:       LD        A,L            ;HL IS SENT TO 8231 STACK
             OUT       (08H),A
             LD        A,H
             OUT       (08H),A
             LD        A,E            ;DE IS SENT TO 8231 STACK
             OUT       (08H),A
             LD        A,D
             OUT       (08H),A
             RET
;
;THIS ROUTINE INPUTS THE RESULT FROM 8231 APU: HL<--- TOS OF 8231
;
INOP:        IN        A,(08H)        ;HL<--- TOP OF STACK
             LD        H,A
             IN        A,(08H)
             LD        L,A
             RET
;
;THIS ROUTINE CONVERTS THE DATA FROM HEXADECIMAL TO DECIMAL AND DISPLAY THEM.
;
DISPLAY:     LD        IY,1600H           ;POINTER<--- 1400H (DATA LOCATION)
             LD        A,0DFH             ;A<--- AMOUNT OF MEMORY AVAILABLE
             LD        (N),A              ;SAVE THE INFO AT COUNTER N
             LD        A,00H
             LD        (NS),A             ;NS<--- 0
DLOOP:       LD        L,(IY)             ;L<--- DATA (LOW)
             INC       IY
             LD        H,(IY)             ;H<--- DATA (HIGH)
             INC       IY
             BIT       7,H                ;CHECK MSB (IS THE NUMBER NEGATIVE?)
             JR        Z,POS              ;IF IT IS POSITIVE, GO TO POS
             LD        A,H                ;IF NOT,A<--- H(DATA-HIGH)
             CPL                          ;COMPLEMENT MSBYTE
             LD        H,A                ;PUT IT BACK TO H
             LD        A,L                ;A<---L (DATA-LOW)
             NEG                          ;A<--- 0-A (2'S COMPLEMENT)
             LD        L,A                ;PUT IT BACK TO L
             LD        A,2DH              ;A<--- ASCII FOR - SIGN
             CALL      ECHO               ;DISPLAYS - SIGN
             JR        CONVERT            ;GO TO CONVERT
POS:         LD        B,01H              ;B<---01H, TO PROVIDE ONE SPACE
             CALL      SPACES
CONVERT:     LD        (NUMBER),HL        ;NUMBER<---DATA TO BE CONVERTED
             LD        DE,0AH             ;DE<--- 10 (BASE FOR CONVERTION)
             CALL      DIVIDE             ;HL<--- NUMBER/10
             LD        (QUOT),HL          ;QUOTIENT<--- NUMBER/10
             CALL      MULTIPLY           ;HL<---QUOTIENT*10
             EX        DE,HL              ;DE<---QUOT*10
             LD        HL,(NUMBER)        ;HL<---NUMBER(DIVIDEND)
             CALL      SUBTRACT           ;HL<---NUMBER-QUOT*10=REMAINDER
             LD        (ONES),HL          ;ONES<---REMAINDER
             CALL      LONGDIV            ;HL<---  NEXT REMAINDER (TENS)
             LD        (TENS),HL          ;TENS<--- REMAINDER
             CALL      LONGDIV
             LD        (HUNDREDS),HL      ;HUNDREDS<---REMAINDER
             CALL      LONGDIV
             LD        (THOUS),HL         ;THOUSANDS<---REMAINDER
             CALL      LONGDIV
             LD        (TTHOUS),HL        ;TENTHOUSANDS<--- REMAINDER
             LD        A,(TTHOUS)         ;A<--- TTHOUS
             CALL      ASCONV             ;CONVERT A TO ASCII AND DISPLAY
             LD        A,(THOUS)          ;SAME THING WITH THE 5 DIGITS
             CALL      ASCONV
             LD        A,(HUNDREDS)
```

```
               CALL      ASCONV
               LD        A,(TENS)
               CALL      ASCONV
               LD        A,(ONES)
               CALL      ASCCNV
               LD        B,02H              ;B<---02H TO PROVIDE 2 SPACES
               CALL      SPACES             ;PUT TWO SPACES BETWEEN NUMBERS
               LD        (TEMPIY),IY        ;STORE POINTER
               LD        A,(TEMPIY)         ;A<--- POINTER.LOW
               AND       OFH                ;CHECK LSB OF THE POINTER
               CP        0                  ;COMPARE WITH ZERO. IF IT IS NOT
               JP        NZ,DLOOP           ;0, THIS ROW IS NOT COMPLETED,REPEAT
               CALL      SCRLF              ;IF IT IS, GO TO THE NEXT LINE
               LD        A,(N)              ;A<--- COUNTER (MEMORY SPACE)
               DEC       A
               LD        (N),A              ;N<--- N-1
               CP        0                  ;END OF MEMORY?
               JR        Z,END              ;IF YES,GO TO END.
               LD        A,(NS)             ;A<--- NS (DISPLAY CONTROL)
               INC       A
               LD        (NS),A             ;NS<--- NS+1
               CP        0AH                ;IS NS=10 ?(IF 10 RUNS HAS ELAPSED)
               JP        NZ,DLOOP           ;IF NOT, GO TO DLOOP
WAIT:          IN        A,(RSSTAT)         ;CHECK THE KEYBOARD INPUT. IF NO KEY
               AND       2                  ;WAS HIT, WAIT.
               JR        Z,WAIT             ;ELSE, RESET AND REPEAT PROCESS
               IN        A,(RECV)
               LD        A,0                ;A<--- 0
               LD        (NS),A             ;SET NS TO ZERO AGAIN
               JP        DLOOP
END:           RET
;
;THIS SUBROUTINE COMPUTES THE LONG DIVISION FOR THE SUBROUTINE DISPLAY
;
LONGDIV:       LD        DE,0AH             ;DE<--- BASE FOR CONVERSION
               LD        HL,(QUOT)          ;HL<--- QUOTIENT
               LD        (NUMBER),HL        ;SAVE NEXT DIVIDEND(LAST QUOTIENT)
               CALL      DIVIDE             ;HL<---QUOTIENT/10
               LD        (QUOT),HL          ;SAVE NEXT QUOTIENT
               CALL      MULTIPLY           ;HL<--- QUOT*10
               EX        DE,HL              ;DE<--- QUOT*10
               LD        HL,(NUMBER)        ;HL<--- NEW DIVIDEND
               CALL      SUBTRACT           ;HL<---DIVIDEND-QUOT*10=REMAINDER
               RET
;
;THIS BLOCK TRANSFERS THE ROUTINE MODEL FROM EPROM TO RAM, ADDRESS 0A00
;
TRANSFER:      LD        HL,MODEL        ;HL<---SOURCE ADDRESS
               LD        DE,1000H        ;DE<---DESTINATION ADDRESS
               LD        BC,03A0H        ;BC<---BLOCK SIZE
               LDIR                      ;TRANSFER,INCREMENT HL AND DE ...
                                         ;DECREMENT BC,IF IS NOT ZERO ...
                                         ;REPEAT PROCESS.
               RET
               DS        1
               END
```

APPENDIX B
MONITOR

1. MONITOR FEATURES

The monitor is a program that provides all the software support for the microprocessor. It works as a supervisor of the system. When the reset switch is hit, the monitor takes over. The program counter is loaded with address zero and the monitor starts performing the initialization routines. The stack pointer is initialized, the serial port (video terminal communication) is set up and some messages are sent to the screen. After that the monitor (program "Main") waits for a command from the keyboard.

The first message sent to the screen is a presentation of the system and the second one asks for a command (E, D or H). Taking the first choice, that is, typing"E", forces the program counter to be loaded with the program Model address and the program will be executed. If "D" is chosen, the results of the last run will be displayed on the screen (program counter is loaded with routine Display address) .If key H (Help) is typed, a list of features provided by the operational system is presented on the screen. All the possible utilizations of the monitor are included in the mentioned list. These features will be detailed in the next paragraphs.

The "List" command permits us to list a portion of the memory. For instance, to look at the memory from addresses 1000H to 1100H the appropriate command is: L1000,1100 < ret > .

The "change" command allows us to make changes in the memories (RAM). Thus, even small routines can be written in the spaces available. It is a powerful tool for debugging programs. For instance, to change data between addresses 15B0H and 15BFH the command should be : C15B0,15BF < ret > . After this command the data contained at memory location 15B0H will be displayed and the cursor will be at the first nibble of this data. To change the actual data, just type the new one. To skip this address, type return.

The "Go" command permits us to run a program that is already in the RAM. Suppose a program was written in the example just mentioned between the addresses 15B0H and 15BFH. The command to run this program is the following: G15B0,15C0 < ret > . Notice that the end address is one address above the last instruction location. The program will run and the CPU registers will be displayed at the end. So, if the program has loaded some results into these registers, they will be available. Also, all the memories addresses can be checked by using the "List" command. The end address of the "Go" command is called "breakpoint" and can be any address inside the program. This allows a valuable debugging process of the program.

The "Registers" command is used to look at the CPU registers (on the screen). The command is issued by typing : < R >.

The "Transfer" command is used to transfer the program "Model" from the EPROM to the RAM (address 1000H). It is issued by typing < T >. Once the program is in the RAM it can be modified by using the "Change" command and can be debugged and executed by using the "Go" command.

When the program "Model" is transfered to the RAM it can be executed by typing < M >. In order to run the program "Model" in the RAM it is necessary to change the address of the last instruction of the main program. This instruction is a jump to the beginning (JP PLOOP) of the program that is now at a different address (the original address is in the EPROM). So, we have to find out the new address of "PLOOP" in the RAM using the PRN file of program "Model" and then change the instruction "JP PLOOP".

The "Display" command is a feature that permits the presentation of the variables of program "Model" on the screen after the execution of the program. The variables are stored in the data memory every time the program is executed in the EPROM or RAM. This command is issued by typing < D >. The variables to be presented on the screen are : Error, XDOT, XDOTE, CDDOT, CM, CS and NN (number of loops). After the command the first ten rows are shown. To get the next ten rows, hit any key.

2. MONITOR PROGRAMS

In this section the programs that belong to the monitor are presented. They were developed during the EC-3800 course and modified a little bit to be used in this thesis.

```
 .Z80
EXTRN           COMMAND,ERRMSG,MESSAGE,MONMSG,SCRLF,TRAP30,TRANMSG,BOUT,TYPMSG
PUBLIC          BKPT,BS,BUFFIN,CHAR,CR,EADDRESS,ESC,FALSE,FLAG,FWDARW
PUBLIC          HEXBUF,LF,MONLOOP,OPCODE,RECV,RSSTAT,RSTART30
PUBLIC          SPACE,SADDRESS,TEMP,TRUE,XMIT
PUBLIC          STAX,AFREG,BCREG,DEREG,HLREG,AFALT,BCALT,DEALT,HLALT
PUBLIC          PCREG,SPREG,IXREG,IYREG,V,CS,MFLAG,NN,KMFLAG,NS
PUBLIC          MONITOR,MODVAR,R,CM,CDOT,CDDOT,CDM1,CDDM1,KM,COUNT,CSM1,CSM2,N
PUBLIC          NUMBER,QUOT,ONES,TENS,HUNDREDS,THOUS,TTHOUS,TEMPIY,CMCF,CF
                DSEG
BKPT:           DS      2       ;BREAKPOINT ADDRESS
BUFFIN:         DS      0FFH    ;INPUT BUFFER
CHAR:           DS      1       ;STORE CRT CHARACTER HERE
EADDRESS:       DS      2       ;END ADDRESS BUFFER
FLAG:           DS      1       ;BOOLEAN FLAG
HEXBUF:         DS      2       ;HEX BUFFER
OPCODE:         DS      1       ;CONTENTS OF BREAKPOINT LOCATION
SADDRESS:       DS      2       ;START ADDRESS BUFFER
STAX:           DS      18H     ;REG STORAGE
MODVAR:         DS      30H     ;MODEL VARIABLE STORAGE
MFLAG:          DS      1       ;MODEL FLAG FOR SUBROUTINE FACTOR
NS:             DS      1       ;COUNTER USED TO CONTROL STORAGE
KMFLAG:         DS      1       ;FLAG USED IN WALG
TEMP:           DS      2       ;TEMP VARIABLE
RECV            EQU     D4H     ;RS-232 INPUT PORT
XMIT            EQU     D4H     ;RS-232 OUTPUT PORT
RSSTAT          EQU     D5H     ;RS-232 STATUS PORT
RSMODE          EQU     D6H
RSCMD           EQU     D7H
PCREG           EQU     STAX    ;PC LOCATION IN STAX
SPREG           EQU     STAX+2
IYREG           EQU     STAX+4  ;IY LOCATION IN STAX
IXREG           EQU     STAX+6  ;IX LOCATION IN STAX
HLALT           EQU     STAX+8  ;HL' LOCATION IN STAX
DEALT           EQU     STAX+0AH        ;DE' LOCATION IN STAX
BCALT           EQU     STAX+0CH        ;BC' LOCATION IN STAX
AFALT           EQU     STAX+0EH        ;AF' LOCATION IN STAX
HLREG           EQU     STAX+10H        ;HL LOCATION IN STAX
DEREG           EQU     STAX+12H        ;DE LOCATION IN STAX
BCREG           EQU     STAX+14H        ;BC LOCATION IN STAX
AFREG           EQU     STAX+16H        ;AF LOCATION IN STAX
R               EQU     MODVAR          ;R (STEP INPUT) LOCATION IN MODVAR
CM              EQU     MODVAR+2        ;CM (MODEL OUTPUT) LOCATION
CMCF            EQU     MODVAR+4        ;CM * CORRECTION FACTOR
CDOT            EQU     MODVAR+6        ;CDOT (DERIVATIVE OF CM) LOCATION
CDDOT           EQU     MODVAR+8        ;CDDOT (ACCELERATION)
CDM1            EQU     MODVAR+0AH      ;PREVIOUS VALUE OF VELOCITY
CDDM1           EQU     MODVAR+0CH      ;PREVIOUS VALUE OF ACCELERATION
CSM1            EQU     MODVAR+0EH      ;PREVIOUS VALUE OF PLANT OUTPUT
CSM2            EQU     MODVAR+10H      ;(T-2) VALUE OF OUTPUT PLANT
N               EQU     MODVAR+12H      ;COUNTER
V               EQU     MODVAR+14H      ;VOLTAGE AT THE LIMITER OUTPUT
KM              EQU     MODVAR+16H      ;MOTOR GAIN CONSTANT
COUNT           EQU     MODVAR+18H      ;COUNTER
CS              EQU     MODVAR+1AH      ;ANALOG OUTPUT
NUMBER          EQU     MODVAR+1CH      ;DATA TO BE DISPLAYED
QUOT            EQU     MODVAR+1EH      ;QUOTIENT
ONES            EQU     MODVAR+20H      ;LSDIGIT OF A DECIMAL NUMBER
TENS            EQU     MODVAR+22H      ;SECOND DIGIT OF A DECIMAL NUMBER
HUNDREDS        EQU     MODVAR+24H      ;THIRD DIGIT
THOUS           EQU     MODVAR+26H      ;FOURTH DIGIT
TTHOUS          EQU     MODVAR+28H      ;FIFTH DIGIT OF A DECIMAL NUMBER
TEMPIY          EQU     MODVAR+2AH      ;TEMPORARY STORAGE OF IY REG.
CF              EQU     MODVAR+2CH      ;CORRECTOR FACTOR
NN              EQU     MODVAR+2EH      ;COUNTER FOR WIKSTROM ALGORITHM
RSTART30        EQU     0F7H            ;OPCODE FOR RST30
FALSE           EQU     0       ;BOOLEAN VARIABLE
TRUE            EQU     0FFH    ;BOOLEAN VARIABLE
BS              EQU     8       ;ASCII BACKSPACE
```

```
FWDARW          EQU     0CH     ;ASCII FOREWARD ARROW
ESC             EQU     1BH     ;ASCII ESCAPE
SPACE           EQU     20H     ;ASCII SPACE
CR              EQU     0DH     ;ASCII CARRIAGE RETURN
LF              EQU     0AH     ;ASCII LINE FEED
;
;       RS-232 PORT CONFIGURATION WORDS
;
MR1             EQU     0CEH
MR2             EQU     7DH
CMD             EQU     5
;
                CSEG
RESET:          LD      SP,1600H
                JP      MONINIT
;
                ORG     30H
RST30:          JP      TRAP30
;
                ORG     38H
INTM1:          JP      09C0H
;
                ORG     66H
NMINT:          JP      09A0H
;
;
;        MONINIT PUSHES ALL THE REGISTERS ONTO THE STACK BEFORE
;        ENTERING THE MONITOR
                ORG     100H
MONINIT:        LD      A,MR1
                OUT     (RSMODE),A
                LD      A,MR2
                OUT     (RSMODE),A
                LD      A,CMD
                OUT     (RSCMD),A
                CALL    SCRLF           ;MOVE CURSOR TO NEXT LINE
                LD      IX,MONMSG       ;SET PTR TO MON MESSAGE
                CALL    MESSAGE         ;PRINT"HI ROBERTO, I AM READY!"
                CALL    SCRLF           ;MOVE CURSOR TO NEXT LINE
                LD      IX,TYPMSG       ;SET POINTER TO TYPE MESSAGE
                CALL    MESSAGE         ;PRINT"TYPE ...E,D,T ..."
                CALL    SCRLF
;
MONLOOP:        CALL    MONITOR         ;INVOKE MONITOR
                JR      MONLOOP         ;LOOP FOREVER
;
MONITOR:        LD      (IXREG),IX      ;SAVE IX AT MONITOR ENTRY
                POP     IX              ;GET PC AT MONITOR ENTRY
                LD      (PCREG),IX      ;STORE PC IN STAX+PCDIS
                LD      (SPREG),SP      ;SAVE SP AT MONITOR ENTRY
                LD      (IYREG),IY      ;SAVE IY AT MONITOR ENTRY
                PUSH    AF              ;PUSH A & F
                POP     IX              ;GET A& F
                LD      (AFREG),IX      ;STORE AF
                LD      (BCREG),BC      ;STORE BC
                LD      (DEREG),DE      ;STORE DE
                LD      (HLREG),HL      ;STORE HL
                EX      AF,AF'
                EXX
                PUSH    AF              ;PUSH A' & F'
                POP     IX              ;IX <-- AF'
                LD      (AFALT),IX      ;STORE AF'
                LD      (BCALT),BC      ;STORE BC'
                LD      (DEALT),DE      ;STORE DE'
                LD      (HLALT),HL      ;STORE HL'
                IN      A,(RSSTAT)      ;GET CONSOLE STATUS
                AND     2               ;IS A CHAR READY
                CALL    NZ,COMMAND      ;GO TO COMMAND DECODER
                LD      IX,(AFALT)      ;RESTORE AF'
                PUSH    IX
                POP     AF
                LD      BC,(BCALT)      ;RESTORE BC'
                LD      DE,(DEALT)      ;RESTORE DE'
                LD      HL,(HLALT)      ;RESTORE HL'
                EX      AF,AF'          ;RESTORE ALL ALT REGS
                EXX
                LD      HL,(HLREG)      ;RESTORE HL
                LD      DE,(DEREG)      ;RESTORE DE
                LD      BC,(BCREG)      ;RESTORE BC
                LD      IX,(AFREG)      ;RESTORE AF
                PUSH    IX
                POP     AF
                LD      IY,(IYREG)      ;RESTORE IY
                LD      SP,(SPREG)      ;RESTORE SP
                LD      IX,(PCREG)      ;RESTORE PC
                PUSH    IX
                LD      IX,(IXREG)      ;RESTORE IX

                RET
;
;
                DS      1
                END


;       THIS PROGRAM DISPLAYS THE CONTENTS OF MEMORY FROM THE
;       STARTING ADDRESS TO THE END ADDRESS
;
 .Z80
PUBLIC          GO,LIST,REG,REGDISP,GHEXERR,DIS,TRF,MOD,HELP
EXTRN           BACKSP,BKPT,BOUT,BUFFIN,CHAR,CHGREGS,COMMA,CR
EXTRN           EADDRESS,EAMSG,ECHO,ERRMSG,ESC,FALSE,FLAG,FWDARW
EXTRN           GETADDR,GETCHAR,GETSTRIN,HEXCNV,HEXCONV,HEXMSG,LINENO,MESSAGE
EXTRN           OPCODE,REGMSG,RSTART30,SADDRESS,SAMSG,SCRLF,SCROLL,SPACES
EXTRN           TEMP,TRUE,DISPLAY,TRANSFER,MODEL,DISMSG,TRANMSG,MODMSG
EXTRN           STAX,SPREG,PCREG,AFREG,IXREG,IYREG,AFALT,H1,H2,H3,H4,H5,H6
EXTRN           H7,H8
;
REGLENGHT       EQU     8               ;LENGHT OF REGLIST
;
GHEXERR:        LD      B,4
                CALL    SPACES
                LD      IX,HEXMSG       ;LOAD HEX CONVERSION MSG
                CALL    MESSAGE
                CALL    SCRLF
GO:             LD      B,2             ;SET 2 SPACES
                CALL    SPACES          ;PRINT 2 SPACES
                CALL    GETSTRIN        ;GET CMD STRING FOR GO
                CALL    HEXCONV         ;CONVERT ASCII TO HEX
                JR      C,GHEXERR       ;IF CARRY IS SET DISPLAY ERROR MSG
                LD      (PCREG),DE      ;ENTER START ADDRESS
                XOR     A               ;CLEAR A
                CP      (IX)            ;IS THERE A BREAKPOINT
                JR      Z,GOEXIT        ;NO, EXIT
                LD      A,COMMA         ;A <-- COMMA
                CP      (HL)            ;IS CHARACTER A COMMA ?
                JR      NZ,GOEXIT       ;IF IT IS NOT, EXIT
                INC     HL              ;ADJUST PTR TO NEXT CHAR
                DEC     (IX)            ;ADJUST CHAR COUNT
                CALL    HEXCNV          ;CONVERT ASCII TO HEX
                JR      C,GHEXERR       ;IF CARRY IS SET DISPLAY ERROR MSG
                LD      A,(DE)          ;GET CONTENTS OF BREAKPT
                LD      (OPCODE),A      ;AND SAVE IN OPCODE
                LD      (BKPT),DE       ;SAVE BREAKPOINT
                LD      A,RSTART30      ;GET OPCODE FOR RST30
                LD      (DE),A          ;INJECT RST30 OPCODE
GOEXIT:         CALL    SCRLF
                RET
```

```
;
LHEXERR:        LD      B,4
                CALL    SPACES
                LD      IX,HEXMSG           ;LOAD HEX CONVERSION MSG
                CALL    MESSAGE
                CALL    SCRLF
LIST:           LD      B,2                 ;SET 2 SPACES
                CALL    SPACES              ;PRINT 2 SPACES
                CALL    GETSTRIN            ;GET CMD STRING FOR LIST
                CALL    HEXCONV             ;CONVERT ASCII ADDRESS
                JR      C,LHEXERR           ;IF CARRY IS SET DISPLAY ERROR MSG
                LD      (SADDRESS),DE       ;START ADDRESS <-- DE
                XOR     A                   ;CLEAR A
                CP      (IX)                ;IS THERE AN END ADDRESS
                JR      Z,LOADEND           ;NO, DEFAULT TO 0
                INC     HL                  ;ADJUST PTR TO NEXT CHAR
                DEC     (IX)                ;ADJUST CHAR COUNT
LOADEND:        CALL    HEXCNV              ;CONVERT ASCII ADDRESS
                JR      C,LHEXERR           ;IF CARRY IS SET DISPLAY ERROR MSG
                LD      (EADDRESS),DE       ;END ADDRESS <-- DE
                CALL    SCRLF
NEWLINE:        CALL    LINENO              ;DISPLAY ADDRESS
                LD      HL,(SADDRESS)       ;GET MEMORY POINTER
GETABYTE:       LD      A,(HL)              ;GET MEMORY BYTE
                CALL    BOUT                ;DISPLAY MEMORY BYTE
                LD      A,(FLAG)
                CP      TRUE                ;IS THE CHANGE FLAG SET
                CALL    Z,CHANGE            ;YES, CHANGE A BYTE
                LD      B,2                 ;SETUP FOR 2 SPACES
                CALL    SPACES              ;PRINT 2 SPACES
                LD      DE,(EADDRESS)       ;DE <-- END ADDRESS
                XOR     A                   ;CLEAR CARRY
                SBC     HL,DE               ;IS START => END
                JR      NC,LISTEXIT         ;NO, EXIT
                LD      HL,(SADDRESS)       ;GET MEMORY POINTER
                INC     HL                  ;INCREMENT START ADDRESS
                LD      (SADDRESS),HL
                LD      A,L                 ;GET SADDRESS.LOW
                AND     0FH                 ;IS THIS A NEW LINE
                JR      NZ,GETABYTE         ;NO, GET A NEW BYTE
                CALL    SCRLF               ;MOVE CURSOR TO NEW LINE
                CALL    SCROLL              ;START & STOP SCROLLING
                LD      A,(CHAR)            ;GET SCROLL CHAR
                CP      ESC                 ;IS IT AN ESCAPE
                JR      Z,LISTEXIT          ;YES, EXIT
                JR      NEWLINE             ;START A NEWLINE
LISTEXIT:       CALL    SCRLF               ;NO, ADJUST CURSOR & EXIT
                RET

;
CHANGE:         LD      B,2                 ;SETUP FOR 2 BACK SPACES
CHGAGIN:        CALL    BACKSP              ;BACK SPACE 2 SPACES
                CALL    GETSTRIN            ;GET ANY NEW CHARACTERS
                LD      A,(BUFFIN)          ;GET STRING LENGTH
                CP      0                   ;IS STRING LENGTH = 0
                JR      Z,NOENTRY
                LD      B,A
                CP      1                   ;IS STRING LENGTH <2
                JR      Z,CHGAGIN           ;YES, DO IT AGAIN
                LD      (TEMP),A            ;SAVE STRING LENGTH
                CALL    HEXCONV             ;CONVERT BYTE TO HEX
                LD      HL,(SADDRESS)       ;RESTORE HL PTR
                LD      (HL),E              ;STORE CHAR IN BYTE
                LD      A,(TEMP)            ;SET CURSOR RESTORE BASE
                NEG                         ;NEGATE STRING LENGTH
NOENTRY:        ADD     A,2                 ;ADD 2 TO RESTORE BASE
                JR      Z,CHANGEX           ;IF 0 ADJUST EXIT
                LD      B,A
                CALL    P,SPACES            ;IF PLUS RESTORE CURSOR
```

```
CHANGEX:        LD      HL,(SADDRESS)   ;RESTORE HL BEFORE RETURN
                RET
;
REG:            CALL    SCRLF
                LD      IX,REGMSG       ;SETUP REG A MESSAGE
                CALL    REGDISP         ;DISPLAY REGISTERS
                LD      IX,CHGREGS      ;SETUP REG CHANGE MESSAGE
                CALL    MESSAGE         ;"ENTER REG TO CHANGE"
                CALL    SCRLF
                CALL    GETCHAR         ;GET SELECTED REGISTER
                CALL    ECHO            ;ECHO REG NAME TO CRT
                LD      A,(CHAR)
                LD      (TEMP),A
                CP      CR              ;NO CHANGE?
                JR      Z,REGEXIT       ;YES, EXIT
                CP      "S"             ;CHANGE SP?
                JR      Z,CHGSP         ;YES, JUMP CHGSP
                CP      "P"             ;CHANGE PC?
                JR      Z,CHGPC         ;YES, JUMP CHGPC
                CP      "X"             ;CHANGE IX?
                JR      Z,CHGIX         ;YES, JUMP CHGIX
                CP      "Y"             ;CHANGE IY?
                JR      Z,CHGIY         ;YES, JUMP CHGIY
                CALL    GETCHAR         ;GET NEXT CHAR IN CMD
                CALL    ECHO
                LD      A,(CHAR)
                CP      "'"             ;IS REG AN ALTERNATE
                JR      Z,CHGALT        ;YES, CHANGE ALT REG SET
                CP      CR              ;END OF CMD?
                JR      NZ,REGERR       ;NO, JUMP REGERR
CHGREG:         LD      HL,AFREG+1      ;GET PTR TO REGS ON STAX
                JR      LOADLIST
CHGALT:         LD      HL,AFALT+1      ;GET PTR TO REGS ON STAX
LOADLIST:       LD      IX,REGLIST      ;GET IX TO "AFBZDZHZZ"
                LD      B,REGLENGHT-1   ;SET REGLIST COUNT
                LD      A,(TEMP)        ;RETRIEVE REG NAME
REGSCAN:        CP      (IX)            ;IS REG = SELECTED REG
                JR      Z,REGCONT       ;YES, OUTPUT CONTENTS
                INC     IX              ;POINT TO NEXT REG
                DEC     HL              ;POINT TO NEXT REG
                DJNZ    REGSCAN         ;GET NEXT REGLIST
                LD      IX,ERRMSG       ;REG NOT FOUND GET ERRMSG
                CALL    MESSAGE         ;"ERROR RE-ENTER"
REGEXIT:        CALL    SCRLF
                RET
REGERR:         LD      IX,ERRMSG
                CALL    MESSAGE
                RET
;
CHGSP:          LD      HL,SPREG+1      ;GET SP AT MON ENTRY
                JR      REGCONT         ;GET NEW CONTENTS
CHGPC:          LD      HL,PCREG+1      ;GET PC AT MON ENTRY
                JR      REGCONT         ;GET NEW CONTENTS
CHGIX:          LD      HL,IXREG+1      ;GET IX AT MON ENTRY
                JR      REGCONT         ;GET NEW CONTENTS
CHGIY:          LD      HL,IYREG+1      ;GET IY AT MON ENTRY
REGCONT:        PUSH    HL              ;SAVE HL
                LD      B,4
                CALL    SPACES          ;PRINT 4 SPACES
                CALL    GETSTRIN        ;GET NEW REG CONTENTS
                CALL    HEXCONV         ;CONVERT CONTENTS TO HEX
                POP     HL              ;RESTORE HL
                LD      A,(TEMP)
                CP      "A"             ;IS REG A?
                JR      Z,AORF          ;YES, GO TO AORF
                CP      "F"             ;IS REG F?
                JR      Z,AORF          ;YES, GO TO AORF
                LD      A,D             ;GET HI BYTE OF HEXBUF
                LD      (HL),A          ;LOAD REG PAIR HI BYTE
                DEC     HL              ;POINT TO LOW REG PAIR
AORF:           LD      A,E             ;GET LO BYTE OF HEXBUF
                LD      (HL),A          ;LOAD REG PAIR LOW BYTE
                CALL    SCRLF           ;OUTPUT CR AND LF
                JR      REGEXIT         ;EXIT
;
REGDISP:        LD      HL,AFREG+1      ;POINT TO A & F IN STAX
                LD      C,2             ;SET LOOP FOR A AND F
                CALL    ONEREG          ;DISPLAY A & F
                LD      C,3             ;SET FOR 3 REGS
                CALL    REGPAIR         ;DISPLAY BC,DE, & HL
                LD      HL,IXREG+1      ;POINT TO IXREG IN STAX
                LD      C,2             ;SET FOR 3 REGS
                CALL    REGPAIR         ;DISPLAY BC,DE, & HL
                CALL    SCRLF
                LD      HL,AFALT+1      ;POINT TO A' & F' IN STAX
                LD      C,2             ;SET LOOP FOR A' AND F'
                CALL    ONEREG          ;DISPLAY A' & F'
                LD      C,3             ;SET FOR 3 REGS
                CALL    REGPAIR         ;DISPLAY BC',DE', & HL'
                LD      HL,SPREG+1      ;POITN TO SP & PC
                LD      C,2             ;SET FOR 2 REGS
                CALL    REGPAIR         ;DISPLAY SP & PC
                CALL    SCRLF           ;GENERATE CR & LF
                RET
;
ONEREG:         CALL    MESSAGE         ;"AF "
                CALL    REGDUMP         ;DISPLAY REG CONTENTS
                INC     IX              ;POINT TO NEXT MSG
                DEC     C               ;DEC REG LOOP COUNTER
                JR      NZ,ONEREG       ;MORE REGS GO TO ONEREG
                RET
;
REGPAIR:        CALL    MESSAGE         ;"HL   "
                CALL    REGDUMP         ;DISPLAY REG CONTENTS
                CALL    REGDUMP         ;DISPLAY REG CONTENTS
                INC     IX              ;POINT TO NEXT MSG
                DEC     C               ;DEC REG LOOP COUNTER
                JR      NZ,REGPAIR      ;MORE REGS GO TO REGPAIR
                RET
;
REGDUMP:        LD      A,(HL)          ;GET REG
                CALL    BOUT            ;OUTPUT REG TO CRT
                DEC     HL              ;POINT TO NEXT REG
                RET
DIS:            CALL    SCRLF
                LD      IX,DISMSG       ;IX<--- MESSAGE ADDRESS
                CALL    MESSAGE         ;" DISPLAY THE RESULTS"
                CALL    SCRLF
                CALL    DISPLAY         ;DISPLAY RESULTS ON SCREEN
                RET
TRF:            CALL    SCRLF
                LD      IX,TRANMSG      ;IX<--- MESSAGE ADDRESS
                CALL    MESSAGE         ;"TRANSFER MODEL TO RAM"
                CALL    SCRLF
                CALL    TRANSFER        ;TRANSFER MODEL TO RAM (1000)
                RET
MOD:            CALL    SCRLF
                LD      IX,MODMSG       ;IX<--- MESSAGE ADDRESS
                CALL    MESSAGE         ;" RUN YOUR MODEL"
                CALL    SCRLF
                CALL    MODEL           ;RUN MODEL
                RET
HELP:           CALL    SCRLF
                LD      IX,H1
                CALL    MESSAGE
                CALL    SCRLF
                CALL    SCRLF
                LD      IX,H2
                CALL    MESSAGE
                CALL    SCRLF
                CALL    SCRLF
                LD      IX,H3
                CALL    MESSAGE
                CALL    SCRLF
                CALL    SCRLF
                LD      IX,H4
                CALL    MESSAGE
                CALL    SCRLF
                CALL    SCRLF
                LD      IX,H5
                CALL    MESSAGE
                CALL    SCRLF
                CALL    SCRLF
                LD      IX,H6
                CALL    MESSAGE
                CALL    SCRLF
                CALL    SCRLF
                LD      IX,H7
                CALL    MESSAGE
                CALL    SCRLF
                CALL    SCRLF
                LD      IX,H8
                CALL    MESSAGE
                CALL    SCRLF
                CALL    SCRLF
                RET
REGLIST:        DC      "AFBZDZHZZ"
                DS      1
                END
;       THIS PROGRAM DECODES COMMANDS AND INVOKES THE PROPER
;       COMMAND ROUTINES
 .Z80
PUBLIC          COMMAND
EXTRN           CHAR,ECHO,ERRMSG,FALSE,FLAG,GETCHAR,GO,LIST,MESSAGE,HELP
EXTRN           REG,SCRLF,TRUE,DIS,TRF,MOD
;
COMMAND:        CALL    GETCHAR         ;GET A CHAR FROM CRT
                CALL    ECHO            ;ECHO CHAR BACK TO CRT
                LD      A,(CHAR)        ;GET CMD CHAR
                CP      'G'             ;IS CHAR A "G"
                JP      Z,GO            ;YES, EXECUTE CODE
                LD      HL,FLAG
                LD      (HL),TRUE       ;SET CHANGE FLAG
                CP      'C'             ;IS CHAR A "C"
                JP      Z,LIST          ;YES,CHANGE MEMORY
                LD      (HL),FALSE      ;CLEAR CHANGE FLAG
                CP      'L'             ;IS CHAR A "S"
                JP      Z,LIST          ;NO, LIST MEMORY
                CP      'R'             ;IS CHAR AN "R"
                JP      Z,REG           ;YES, DISPLAY REGS
                CP      'D'             ;IS CHARACTER A "D"?
                JP      Z,DIS           ;YES,DISPLAY RESULTS
                CP      'T'             ;IS CHARACTER A "T"?
                JP      Z,TRF           ;TRANSFER MODEL TO RAM
                CP      'E'             ;IS CHARACTER AN "E"?
                JP      Z,MOD           ;RUN MODEL
                CP      'M'             ;IS CHARACTER AN "M" ?
                JP      Z,1000H         ;GO TO RAM, ADDRESS 1000H
                CP      'H'             ;IS CHARACTER AN "H" ?
                JP      Z,HELP          ;IF YES, GO TO HELP (IN LIST)
                LD      IX,ERRMSG       ;GET ERROR MESSAGE PTR
                CALL    MESSAGE         ;PRINT "ERROR RE-ENTER"
                CALL    SCRLF           ;MOVE CURSOR TO NEXT LINE
                RET
                END

;       THIS PROGRAM DECODES COMMANDS AND INVOKES THE PROPER
;       COMMAND ROUTINES
 .Z80
PUBLIC          COMMAND
EXTRN           CHAR,ECHO,ERRMSG,FALSE,FLAG,GETCHAR,GO,LIST,MESSAGE,HELP
EXTRN           REG,SCRLF,TRUE,DIS,TRF,MOD
;
COMMAND:        CALL    GETCHAR         ;GET A CHAR FROM CRT
                CALL    ECHO            ;ECHO CHAR BACK TO CRT
                LD      A,(CHAR)        ;GET CMD CHAR
                CP      'G'             ;IS CHAR A "G"
                JP      Z,GO            ;YES, EXECUTE CODE
                LD      HL,FLAG
                LD      (HL),TRUE       ;SET CHANGE FLAG
                CP      'C'             ;IS CHAR A "C"
                JP      Z,LIST          ;YES,CHANGE MEMORY
                LD      (HL),FALSE      ;CLEAR CHANGE FLAG
                CP      'L'             ;IS CHAR A "S"
                JP      Z,LIST          ;NO, LIST MEMORY
                CP      'R'             ;IS CHAR AN "R"
                JP      Z,REG           ;YES, DISPLAY REGS
                CP      'D'             ;IS CHARACTER A "D"?
                JP      Z,DIS           ;YES,DISPLAY RESULTS
                CP      'T'             ;IS CHARACTER A "T"?
                JP      Z,TRF           ;TRANSFER MODEL TO RAM
                CP      'E'             ;IS CHARACTER AN "E"?
                JP      Z,MOD           ;RUN MODEL
                CP      'M'             ;IS CHARACTER AN "M" ?
                JP      Z,1000H         ;GO TO RAM, ADDRESS 1000H
                CP      'H'             ;IS CHARACTER AN "H" ?
                JP      Z,HELP          ;IF YES, GO TO HELP (IN LIST)
                LD      IX,ERRMSG       ;GET ERROR MESSAGE PTR
                CALL    MESSAGE         ;PRINT "ERROR RE-ENTER"
                CALL    SCRLF           ;MOVE CURSOR TO NEXT LINE
                RET
                END

;       THIS PROGRAM OUTPUTS MESSAGES TO THE CRT SCREEN
;
 .Z80
EXTRN           ECHO
PUBLIC          CHGREGS,EAMSG,ERRMSG,HEXMSG,MESSAGE,MONMSG,REGMSG,SAMSG
PUBLIC          TRANMSG,MODMSG,HELLO,DISMSG,TYPMSG,H1,H2,H3,H4,H5,H6,H7,H8
NULL            EQU     0               ;ASCII NULL SYMBOL
;
MESSAGE:        LD      A,(IX)          ;GET MESSAGE CHAR
                CP      NULL            ;IS CHAR = NULL
                JR      Z,MSGRET        ;YES, RETURN
                CALL    ECHO            ;OUTPUT CHAR TO CONSOLE
                INC     IX              ;INCREMENT MESSAGE PTR
                JR      MESSAGE         ;GET ANOTHER CHARACTER
MSGRET:         RET
CHGREGS:        DC      "ENTER REGISTER YOU WANT TO CHANGE"
                DC      " [A,F,B(BC),D(DE),H(HL),S(SP),P(PC)]"
                DB      NULL
EAMSG:          DC      "END ADDRESS "
                DB      NULL
ERRMSG:         DC      "ERROR RE-ENTER"
                DB      NULL
HEXMSG:         DC      "HEX CONVERSION ERROR...RE-ENTER"
                DB      NULL
MONMSG:         DC      "AUTOADAPTIVE CONTROL FOR A ROBOT ARM"
                DB      NULL
REGMSG:         DB      "A=",NULL,"  F=",NULL,"  BC=",NULL,"  DE="
                DB      NULL,"  HL=",NULL,"  IX=",NULL,"  IY=",NULL
                DB      "A'",NULL,"  F'",NULL,"  BC'",NULL,"  DE'"
                DB      NULL,"  HL'",NULL,"  SP=",NULL,"  PC=",NULL
SAMSG:          DC      "START ADDRESS "
                DB      NULL
TYPMSG:         DC      "TYPE 'E' TO RUN MODEL, "
```

```
                DC      "'D' TO DISPLAY THE RESULTS, "
                DC      "'H' TO GET HELP."
                DB      NULL
TRANMSG:        DC      "TRANSFER MODEL FROM EPROM TO RAM ADDRESS 1000"
                DB      NULL
DISMSG:         DC      "DISPLAY RESULTS"
                DB      NULL
MODMSG:         DC      "RUN MODEL"
                DB      NULL
HELLO:          DC      "PLEASE ENTER ANGLE POSITION. MAXIMUM IS 7F"
                DB      NULL
H1:             DC      "1.TO RUN THE PROGRAM, TYPE <E>.ENTER ANGLE(HEX),"
                DC      "RESET THE INTEGRATORS IN THE ANALOG PLANT AND HIT"
                DC      "<RETURN>.TO STOP THE PROGRAM, PRESS THE RESET SWITCH"
                DB      NULL
H2:             DC      "2.TO DISPLAY THE RESULTS OFF LINE (LAST RUN),TYPE <D>"
                DB      NULL
H3:             DC      "3.TO TRANSFER THE PROGRAM TO RAM (AD. 1000H),TYPE <T>"
                DB      NULL
H4:             DC      "4.TO RUN THE PROGRAM IN THE RAM, TYPE <M>."
                DB      NULL
H5:             DC      "5.TO LIST THE MEMORY FROM ADDRESS XXXX TO YYYY, TYPE:"
                DC      "'L<RET>XXXX,YYYY<RET>'"
                DB      NULL
H6:             DC      "6.TO CHANGE DATA OR ENTER MACHINE LANGUAGE PROGRAM"
                DC      "INTO RAM BETWEEN THE ADRESSES XXXX AND YYYY, TYPE:"
                DC      "'C<RET>XXXX,YYYY<RET>. TYPE <RET> TO BROWSE THE "
                DC      "MEMORY"
                DB      NULL
H7:             DC      "7.TO RUN A PROGRAM IN THE RAM TYPE:G<RET>XXXX,ZZZZ"
                DC      "<RET> WHERE ZZZZ IS THE BREAKPOINT OF THE PROGRAM."
                DC      "THE CPU REGISTERS WILL BE SHOWN ON THE SCREEN."
                DB      NULL
H8:             DC      "8.TO SEE THE CPU REGISTERS, TYPE <R>."
                DB      NULL
                END
;
;THIS PROGRAM CONTAINS THE ROUTINES TO GET CHARACTERS FROM THE CRT
;KEYBOARD AND SEND THEM TO THE SCREEN. FILE IS "CONSOLE"
;
EXTRN           CHAR,REC,RSSTAT,XMIT
PUBLIC          GETCHAR,ECHO
GETCHAR:        IN      A,(RSSTAT)      ;GET CONSOLE STATUS
                AND     2               ;IS A CHAR READY ?
                JR      Z,GETCHAR       ;NO, CHECK AGAIN
                IN      A,(RECV)        ;YES, GET RS232 DATA
                AND     7FH             ;PARITY STRIP
                LD      (CHAR),A        ;STORE INPUT IN CHAR
                RET
ECHO:           LD      E,A             ;TEMP CHAR STORE
CKAGIN:         IN      A,(RSSTAT)      ;GET CONSOLE STATUS
                AND     1               ;IS XMIT READY ?
                JR      Z,CKAGIN        ;NO, CHECK AGAIN
                LD      A,E
                OUT     (XMIT),A        ;YES, SEND DATA OUT
                RET
                DS      1
                END
;

;      THIS PROGRAM CONVERTS THE STRING IN BUFFIN INTO A HEXIDECIMAL NUMBER
;
EXT             BUFFIN,HEXSHIFT,LF,SEVEN,THIRTY
PUBLIC          HEXCONV,HEXCNV,COMMA
;
CLEAR           EQU     00H              ;INITIALIZATION VALUE
COMMA           EQU     2CH              ;ASCII COMMA
HT              EQU     09H              ;ASCII HORIZONTAL TABULATION
DLE             EQU     10H              ;ASCII DATA LINK ESCAPE
```

```
;
HEXCONV:        LD      HL,BUFFIN+1      ;INITIATE BUFFIN PONITER TO BUFFIN(1)
                CALL    HEXCNV           ;CONVERT THE START ADDRESS TO HEX
                RET
;
;           THIS ROUTINE PERFORMS THE ASCII TO HEX CONVERSION
HEXCNV:         LD      DE,CLEAR         ;CLEAR HEXIDECIMAL REGISTERS
                LD      IX,BUFFIN        ;SET IX REG AS A STRING LENGTH POINTER
                LD      A,(IX)           ;EVALUATE STRING LENGTH
                CP      CLEAR            ;IS THE STRING LENGTH EQUAL TO ZERO
                JR      Z,EMPTY          ;IF LENGTH EQUALS ZERO EXIT ROUTINE
HEXLOOP:        LD      A,(HL)           ;GET CHARACTER FROM BUFFIN
                CP      COMMA            ;IS CHARACTER A COMMA ?
                JR      Z,EMPTY          ;IF IT IS EXIT ROUTINE
                SUB     THIRTY           ;A <-- CHAR -30H
                LD      (HL),A           ;CHAR <-- A REG
                CP      LF               ;IS CHARACTER 0AH ?
                JP      M,ZEROCK         ;IF IS LESS THAN 0AH GO TO ZEROCK
                SUB     SEVEN            ;A <-- CHAR - 07H
                LD      (HL),A           ;CHAR <-- A REG
                CP      HT               ;IS CHARACTER A 09H ?
                JP      M,HEXERR         ;IF IS LESS THAN 09H GO TO HEXERR
                CP      DLE              ;IS CHARACTER A 10H ?
                JP      P,HEXERR         ;IF GREATER THAN 10H GO TO HEXERR
ZEROCK:         CP      CLEAR            ;IS CHARACTER A ZERO ?
                JP      M,HEXERR         ;IF LESS THAN ZERO GO TO HEXERR
                CALL    HEXSHIFT         ;PROVIDE THE SHIFT IN THE HEX REGISTER
                INC     HL               ;INCREMENT BUFFIN POINTER
                DEC     (IX)             ;DECREMENT STRING LENGTH
                XOR     A                ;CLEAR ACCUMULATOR
                CP      (IX)             ;IS THERE MORE CHARACTERS ?
                JP      NZ,HEXLOOP       ;IF YES GET NEXT CHARACTER
EMPTY:          OR      A                ;NO ERROR ,SO CLEAR CARRY
ENDCONV:        RET
HEXERR:         SCF                      ;ERROR ,SO SET CARRY
                JR      ENDCONV          ;EXIT SUBROUTINE
;
                DS 1
                END
;  THIS PROGRAM PROVIDES THE SHIFT OF THE HEX ADDRESS INTO THE HEX REGISTERS
;
PUBLIC          HEXSHIFT
;
HEXSHIFT:       LD      B,04H            ;B <-- 04
                OR      A                ;CLEAR CARRY
SHIFT:          SLA     E                ;SHIFT LEFT E CONTENTS THROUGH CARRY
                RL      D                ;ROTATE LEFT D CONTENTS FROM CARRY
                DJNZ    SHIFT            ;ARE THE FOUR SHIFTS COMPLETED ?
                LD      A,(HL)           ;A <-- CONVERTED CHARACTER
                OR      E                ;SUPERIMPOSE THE CONTENTS OF E AND A
                LD      E,A              ;E <-- A REG
                RET
;
                DS      1
                END
;   THIS PROGRAM GETS A STRING OF CHARACTERS FROM THE CRT THAT WILL BE
;                     CONCATENATED BY CONCAT
;
EXT             BUFFIN,CONCAT,CR,ECHO,GETCHAR
PUBLIC          GETSTRIN
;
ZERO            EQU     00H              ;INITIAL VALUE OF COUNTER
;
GETSTRIN:       LD      C,ZERO           ;COUNTER INITIALIZATION
                LD      HL,BUFFIN        ;INITIATE BUFFIN POINTER
                INC     HL               ;PLACE POINTER INTO BUFFIN(1)
BUILD:          CALL    GETCHAR          ;READ CHARACTER FROM CRT
                CP      CR               ;IS THE CHARACTER A CARRIAGE RETURN ?
                JR      Z,STRINGEX       ;IF IT IS JUMP TO STINGEX
                CALL    ECHO            ;DISPLAY CHARACTER
                CALL    CONCAT          ;CONCATENATE THIS CHAR WITH THE OTHERS
                JR      BUILD           ;GET NEXT CHARACTER
STRINGEX:       LD      A,C             ;A REG <-- NUMBER OF CHAR IN STRING
                LD      (BUFFIN),A      ;STRING LENGTH IS STORED IN BUFFIN(0)
                RET
;
                DS 1
                END
;       THIS PROGRAM DISPLAY THE STARTING ADDRESS OF THE CURRENT
;       LINE
;
 .Z80
PUBLIC          LINENO
EXTRN           BOUT,SADDRESS,SPACES
;
LINENO:         LD      A,(SADDRESS+1)  ;GET MSB CURRENT ADDRESS
                CALL    BOUT            ;DISPLAY HI ADDRESS BYTE
                LD      A,(SADDRESS)    ;GET LSB CURRENT ADDRESS
                CALL    BOUT            ;DISPLAY LOW ADDRESS BYTE
                LD      B,4             ;SETUP FOR 4 SPACES
                CALL    SPACES          ;PRINT 4 SPACES
                RET
                DS      1
                END
;       THIS PROGRAM SENDS OUT A CARRIAGE RETURN AND LINE FEED
;
 .Z80
;
PUBLIC          BACKSP,SCRLF,SCROLL,SPACES
EXTRN           BS,CHAR,CR,ECHO,ESC,GETCHAR,LF,RSSTAT
EXTRN           FWDARW
;
BACKSP:         LD      A,BS            ;A <-- BACK SPACE
                CALL    ECHO            ;SEND SPACE TO CRT
                DJNZ    BACKSP          ;YES, GOTO SPACES
                RET
;
SCRLF:          LD      A,CR            ;A <-- ASCII RETURN
                CALL    ECHO            ;SEND A RETURN TO CRT
                LD      A,LF            ;A <-- ASCII LINE FEED
                CALL    ECHO            ;SEND A LINE FEED TO CRT
                RET
;
SPACES:         LD      A,FWDARW        ;A <-- FOREWARD ARROW
                CALL    ECHO            ;SEND SPACE TO CRT
                DJNZ    SPACES          ;YES, GOTO SPACES
                RET
;
SCROLL:         IN      A,(RSSTAT)      ;GET CONSOLE STATUS
                CP      2               ;IS A CHAR READY
                JR      NZ,SEXIT        ;NO, EXIT
                CALL    GETCHAR         ;YES, GET THE CHARACTER
                LD      A,(CHAR)        ;A <-- CHAR
                CP      ESC             ;IS THE CHAR AN ESCAPE
                JR      Z,SEXIT         ;YES, TERMINATE SCROLL
PAUSECK:        IN      A,(RSSTAT)      ;GET CONSOLE STATUS
                CP      2               ;IS A CHAR READY
                JR      NZ,PAUSECK      ;NO, CONTINUE PAUSE
                CALL    GETCHAR         ;CLEAR REC REG
SEXIT:          RET
                DS      1
                END
;       THIS PROGRAM GETS AN ASCII ADDRESS AND CONVERTS IT TO HEX
;
;
 .Z80
PUBLIC          GETADDR
EXTRN           BUFFIN,GETSTRIN,HEXCONV,MESSAGE,SCRLF
```

```
;
GETADDR:        CALL    SCRLF           ;MOVE CURSOR TO NEXT LINE
                CALL    MESSAGE         ;PRINT ADDRESS MESSAGE
                CALL    GETSTRIN        ;GET ASCII ADDRESS
                CALL    SCRLF           ;MOVE CURSOR TO NEXT LINE
                CALL    HEXCONV         ;CONVERT ADDRESS TO HEX
                JR      C,GETADDR       ;ERROR, GET NEW ADDRESS
ADDREXIT:       RET
                DS      1
                END
;   THIS PROGRAM IS USED BY GETSTRIN TO CONCATENATE THE CHARATER ONTO THE
;             ASCII STRING BEING FORMED IN THE INPUT BUFFER
;
EXT             CHAR,BS
PUBLIC          CONCAT
;
CONCAT:         CP      BS              ;IS THE CHARACTER A BACK SPACE ?
                JR      Z,CORRECT       ;IF IT IS JUMP TO CORRECT
                LD      A,(CHAR)        ;A REG <-- LOADED CHARACTER
                LD      (HL),A          ;STORE CHARACTER IN BUFFIN
                INC     C               ;INCREMENT COUNTER
                INC     HL              ;SET POINTER TO NEXT BUFFIN POSITION
                RET
CORRECT:        LDD                     ;HL <-- HL-1,BC <-- BC-1
                RET
;
        DS      1
        END
;          THIS PROGRAM CONVERTS THE HEX CONTENTS OF A INTO ASCII
;
EXT             TEMP,ECHO
PUBLIC          BOUT,THIRTY,SEVEN,ASCONV
;
MASK            EQU     0FH             ;ASCII MASK
THIRTY          EQU     30H             ;ASCII   30
COLON           EQU     3AH             ;ASCII    :
SEVEN           EQU     07H             ;ASCII    7
;
BOUT:           LD      (TEMP),A        ;MAKE TEMPORARY COPY OF ACCUMULATOR
                SRL     A               ;FIRST SHIFT RIGHT OF A CONTENTS
                SRL     A               ;SECOND SHIFT RIGHT OF A CONTENTS
                SRL     A               ;THIRD SHIFT RIGHT OF A CONTENTS
                SRL     A               ;FOURTH AND LAST SHIFT RIGHT
                CALL    ASCONV          ;CONVERSION OF MS NIBBLE
                LD      A,(TEMP)        ;GET TEMPORARY COPY OF A
                AND     MASK            ;MASK OFF MS NIBBLE
                CALL    ASCONV          ;CONVERSION OF LS NIBBLE
                RET                     ;RETURN TO CALLING ROUTINE
;
; THIS SUBROUTINE WORKS WITH 'BOUT' TO PERFORM THE HEX TO ASCII CONVERSION
;
ASCONV:         ADD     A,THIRTY        ;ADD 30H TO HEX NIBBLE
                CP      COLON           ;COMPARE THE RESULT OF SUM WITH COLON
                JP      M,ASCOUT        ;IF IT IS LESS THEN COLON JP TO ASCOUT
                ADD     A,SEVEN         ;ADD 7 FOR A-F OFF SET
ASCOUT:         CALL    ECHO            ;DISPLAY THE ASCII CHARACTER
                RET                     ;GO BACK TO 'BOUT'
;
                DS      1
                END
;  -         THIS PROGRAM RESTARTS THE MONITOR AFTER A BREAKPOINT
;
EXT             BKPT,OPCODE,REGDISP,REGMSG,MONLOOP,MONITOR
PUBLIC          TRAP30
;
TRAP30:         CALL    MONITOR         ;SAVE REGISTERS
                LD      A,(OPCODE)      ;A <--OPCODE OF BREAKPOINT POSITION
                LD      DE,(BKPT)       ;DE<--ADDRESS OF BREAKPOINT
                LD      (DE),A          ;RESTORE OPCODE IN THE PROGRAM
LD      IX,REGMSG       ;PREPARE TO DISPLAY REGISTERS
CALL    REGDISP         ;DISPLAY REGISTERS
JP      MONLOOP         ;GO TO MONLOOP AT MAIN
RET
DS      1
END
```

APPENDIX C
EQUIPMENT AND OPERATIONAL PROCEDURES

1. EQUIPMENT

The experimental system consists of the following parts:

1. Three protoboards for the microprocessor. extra memories (data memories) and analog plant
2. One video terminal
3. One +5 V / 5 A power supply
4. One -5 V / 300 mA power supply
5. One +12 V / 300 mA power supply
6. One ± 15 V/ 1 A power supply
7. One oscilloscope for measuring the analog output.

2. PROCEDURE

In the actual configuration the objective of the hardware is to receive a commanded input from the keyboard (robot arm angle to be displaced) and provide the corresponding voltage at the plant output. This voltage represents the actual displacement of the robot arm and is monitored by the oscilloscope at the plant output. The commanded input (hexadecimal), when converted to decimal represents a multiple of 2.23 degrees. The minimum input allowed is 1 and the maximum is 7F (hexadecimal). In decimal they represent inputs from 1 to 127 and in degrees they are equivalent to angles between 2.23 to 284.2 dgrees, respectively.

The operational procedure to run the system, in a step by step basis can be described as follows :

1. Turn on the main power, power supplies, monitor and oscilloscope.
2. Connect one channel of the oscilloscope to the plant output (CS). A second channel can be used to observe the plant input (V).
3. Reset the microprocessor by pressing the reset switch. The screen will display three options (execution, display and help). Hit < E > to run the program.
4. The program will ask for the angle input. Type an hexadecimal number less or equal to 7FH (decimal 127). Adjust the oscilloscope scale accordingly. The biggest voltage will be equal to the decimal input multiplied by 0.039 Volts.
5. Reset the integrators in the analog plant. the last integrator switch must be released last to guarantee zero output at the beginning of the program.
6. Type < ret > right after releasing the reset switch.

7. The program will run and the plant output can be observed at the oscilloscope. The voltage on the oscilloscope represents the angle position in radians.

8. Reset the microprocessor.

9. Type < D > on the keyboard to display the results. As explained in Appendix C, the variables will be displayed in eight columns. From left to right they will be : Error, XDOT, XDOTE, CDDOT, CDOT, CM, CS and NN (number of loops accomplished by the program up to that row).

10. To scroll up the rows and look at some more data, hit any key.

11. At the end of the data memory the screen will show an error message to indicate that there is no more data available. To look at the data again, reset the micro and type < D > .

LIST OF REFERENCES

1. Wikstrom, K., *An Adaptive Model Based Disk File Head Positioning Servo System*, Master Thesis, Naval Postgraduate School, Monterey, CA, September 1985.

2. Zaks, R., *Programming the Z80*, Sybex, 1982.

3. Ogata, K., *Modern Control Engineering*, Prentice Hall, 1970.

4. Roberge, J., *Operational Amplifiers: Theory and Practice*, John Wiley & sons, 1975.

5. Advanced Micro Devices Inc., *Am9500 Peripherical Products Interface Guide*, 1980.

6. Tobey, G. E., Graeme, J. G. and Huelsman, L. P., *Operational Amplifiers, Design and Applications*, Burr-Brown Researcher Corporation, 1971.

7. Ozaslan, K., *The Near Minimum Time Control of a Robot Arm*, Master Thesis, Naval Postgraduate School, Monterey, CA, December 1986.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Professor G. J. Thaler, Code 62Tr Department of Electrical & Computer Engineering Naval Postgraduate School Monterey, CA 93943	1
4.	Professor H. A. Titus, Code 62Ti Department of Electrical & Computer Engineering Naval Postgraduate School Monterey, CA 93943	1
5.	Professor Larry W. Abbot, Code 62At Department of Electrical & Computer Engineering Naval Postgraduate School Monterey, CA 93943	
6.	Professor L. W. Chang, Code 69Ck Department of Mechanical Engineering Naval Postgraduate School Monterey, CA 93943	1
7.	Professor D. L. Smith, Code 69Sm Department of Mechanical Engineering Naval Postgraduate School Monterey, CA 93943	1
8.	Professor R. Werneth, Code 69Wh Department of Mechanical Engineering Naval Postgraduate School Monterey, CA 93943	1
9.	Cpt. Paulo Roberto de Souza Centro Tecnico Aeroespacial - IAE - ESB 12225 - S. Jose dos Campos - SP - Brasil	2
10.	Diretor do Centro Tecnico Aeroespacial Centro Tecnico Aeroespacial 12225 - S. Jose dos Campos - SP - Brasil	1
11.	Diretor do Instituto de Atividades Espaciais Centro Tecnico Aeroespacial - IAE 12225 - S. Jose dos Campos - SP - Brasil	1
12.	Chefe da Divisao de Sistemas Belicos Centro Tecnico Aeroespacial - IAE - ESB 12225 - S. Jose dos Campos - SP - Brasil	1
13.	Biblioteca do ITA Centro Tecnico Aeroespacial - ITA 12225 - S. Jose dos Campos - SP - Brasil	1

14.	Diretor do IPD Centro Tecnico Aeroespacial - IPD 12225 - S. Jose dos Campos - SP - Brasil	1
15.	Reitor do ITA Centro Tecnico Aeroespacial - ITA 12225 - S. Jose dos Campos - SP - Brasil	1
16.	LT. Antonio J. Gameiro Marques D.S.I.T. Edificio da Administracao Central de Marinha 1188, Lisboa CODEX - Portugal	1
17.	Kemal Ozaslan Celebidere Sokak NO=23 4 Yenikoy İstanbul-Turkey	1
18.	Nusret Yurutucu 720 Trenton Place Gilroy, CA 95020	1

221170

Thesis
S666235 Souza
c.1 Real time programming
of a robot.

221170

Thesis
S666235 Souza
c.1 Real time programming
of a robot.