To the Graduate Council:

I am submitting herewith a thesis written by Laurence Allan Phillips entitled "Development and Performance of a Microprocessor for Closed-Loop Digital Control." I recommend that it be accepted in partial fulfillment of the requirements for Master of Science, with a major in Electrical Engineering.

Walter L. Green

Walter L. Green, Major Professor

We have read this thesis
and recommend its acceptance:

J M Googe

R.E. Bodenheimer

H M Mathews

Accepted for the Council:

Hilton A. Smith

Vice Chancellor
Graduate Studies and Research

# DEVELOPMENT AND PERFORMANCE OF A MICROPROCESSOR FOR CLOSED-LOOP DIGITAL CONTROL

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Laurence Allan Phillips

March, 1976

1269766

# ACKNOWLEDGMENTS

The author would like to express his sincere appreciation to Dr. Walter L. Green for his patience, encouragement, support and friendship throughout the graduate program.

This thesis is dedicated to the author's parents, Elwyn and Margaret Phillips, and brother, Leon Phillips, without whose faith and guidance it would not have existed.

The author wishes also to thank Ms. Kay Anderson, Ms. Delores Buckner, and Ms. June Wilson for their diligence and care in the typing of the thesis.

A special thanks also to Tennessee Operations of Aluminum Company of America for their willing support in the preparation of the thesis.

# ABSTRACT

This thesis gives the results achieved from constructing a microprocessor for digital control of a feedback system. The computer was constructed using modular plug-in cards purchased from Control Logic, Incorporated. Various additional hardware that was interfaced to the microprocessor was either bought as off-the-shelf items or fabricated from conventional integrated circuits. Software was developed for processing a second-order control algorithm that could be used in either a PID or lead/lag application. A type zero analog system having three poles was selected as a plant for testing and evaluating digital control with the microprocessor. A description of the computer is presented along with the control performance achieved with this system.

TABLE OF CONTENTS


| CHAPTER | | PAGE |
|---|---|---|
| I. | INTRODUCTION | 1 |
| | Background | 1 |
| | Industrial Applications | 3 |
| | Future Trends | 4 |
| II. | MICROPROCESSOR SYSTEM ORGANIZATION | 6 |
| | Basic System Organization | 6 |
| | Central Processing Unit | 8 |
| | Input-Output Structure | 11 |
| | Memory Organization | 15 |
| III. | PERIPHERAL DESIGN | 19 |
| | Signal Servicing | 19 |
| | Clock Scaler | 22 |
| | Multiplier | 24 |
| IV. | DEVELOPMENT AND PROGRAMMING OF CONTROL ALGORITHM | 37 |
| | PID Algorithm Background and Development | 37 |
| | Algorithm Programming | 41 |
| | Results | 48 |
| V. | CONCLUSIONS AND RECOMMENDATIONS | 62 |
| LIST OF REFERENCES | | 65 |
| APPENDIX | | 68 |
| VITA | | 98 |

# LIST OF TABLES


| TABLE | | PAGE |
|---|---|---|
| 4.1. | Comparison of Simulation to Actual Results . . . | 61 |
| A.1. | Decimal Integer to Octal . . . . . | 74 |
| A.2. | Decimal Tenths to Octal . . . . . | 74 |
| A.3. | Decimal Hundredths to Octal . . . . . | 77 |
| A.4. | Decimal Thousandths to Octal . . . . . | 77 |
| A.5. | Microprocessor Instruction Set . . . . . | 80 |
| A.6. | Memory Storage Area (Page Three) . . . . . | 84 |

# LIST OF FIGURES


| FIGURE | | PAGE |
|---|---|---|
| 2.1. | Microprocessor Architecture | 7 |
| 2.2. | CPU State Transition Diagram | 10 |
| 2.3. | CPU Architecture | 12 |
| 2.4. | I/O Signal Timing Diagram | 14 |
| 2.5. | Memory Address Storage | 17 |
| 3.1. | Peripheral Interfacing Structure | 20 |
| 3.2. | ADC Architecture | 21 |
| 3.3. | Interconnection of SHA to ADC | 23 |
| 3.4. | Basic Configuration of Clock Scaler | 25 |
| 3.5. | Frequency Scaler with Factor of Five | 26 |
| 3.6. | Frequency Scaler with Factor of Ten | 28 |
| 3.7. | Program Structure and Timing Characteristics of a Software Multiply Subroutine | 29 |
| 3.8. | Application of the Software Multiply Subroutine to a Digital Algorithm | 31 |
| 3.9. | Iterative Cell Structure of Hardware Multiplier | 34 |
| 3.10. | Shift-and-Conditionally-Add Iterative Cell | 35 |
| 3.11. | 8-Bit Complement Circuitry | 36 |
| 4.1. | PID Block Diagram Representation | 39 |
| 4.2. | Flow Chart of Microprocessor Controller | 44 |
| 4.3. | Microprocessor Controller as Applied to the Plant System | 49 |

FIGURE PAGE

4.4. Dynamic Response of Uncontrolled System for a 2.0 Voltage Step Change . . . . . 51

4.5. Dynamic Response of Controlled System for a 2.0 Voltage Step Change . . . . . 54

4.6. Block Diagram of CSMP Simulation . . . . . 55

4.7. CSMP Program for Uncontrolled System . . . . . 57

4.8. CSMP Program for Controlled System . . . . . 58

4.9. CSMP Dynamic Response of Uncontrolled System for a 2.0 Voltage Step Change . . . . 59

4.10. CSMP Dynamic Response of Controlled System for a 2.0 Voltage Step Change . . . . . 60

A.1. Flow Chart of Subroutine ROLL . . . . . 73

A.2. Flow Chart of Subroutine ALIN . . . . . 78

# CHAPTER I

## INTRODUCTION

Current control system philosophies include the use of digital computers in feedback control systems. The microprocessor offers an economical advantage that has stimulated interest in its application to feedback control systems. This thesis presents the development of a microprocessor control system that is versatile enough to process many different types of control strategies. To illustrate the effectiveness of the system, a Proportional-Integral-Derivative (PID) control algorithm is developed and implemented. The purpose of this type of microprocessor application is to study its advantages and limitations and to make recommendations as to its future use in the field of controls. Before presenting the fundamentals of the microprocessor and its application, a brief history and future development trend of the microprocessor in general is outlined in the following paragraphs.

### Background

In the mid 1960's, the concepts and technologies behind computer designs began making rapid advancements and changes(6). Prior to this time, computers were mainly designed to be used for business and research purposes with emphasis placed upon computational power and memory quantities. Engineers soon realized that size, power

requirements and high-speed performance were often more critical than large-scale, general-purpose, computing capability. Further advances in Medium Scale Integration (MSI) and development of Large Scale Integration (LSI) initiated the introduction of the minicomputer. The price of minicomputers was such as to justify their use as a dedicated control unit yet remain versatile and powerful enough to perform many tasks. Industry began to accept computers as a more efficient means for real-time control.

In the late 1960's, LSI progressed to the point where hundreds of memory locations were available on one small device. Read Only Memory (ROM) complemented Random Access Memory (RAM) and introduced structured micro-programming to small computer systems(6). Programmable Read Only Memory (PROM) allowed reprogramming of permanent memories. The manufacturers of these new memories successfully implemented a compatible LSI processor. Realizing the possibilities for the future of such computing systems, and once again encouraged by present demand for compactness by many industries, the pioneer company (Intel Corporation) built a system of MSI logic devices controlled by the LSI processor.

Among the first of these units was a four-bit, parallel microprocessor called the INTEL 4004. This processor was soon followed by the eight-bit, parallel-structured INTEL 8008 and INTEL 8080. Several companies have added MSI logic to the INTEL LSI chips to market their

own microprocessors, while other companies have subsequently produced LSI chips and processors of their own design(12).

## Industrial Applications

The microprocessor has definite potential applications in industrial control where minicomputers are now used. Minicomputer applications are often justified only when used to control either multiple-functions or a single large-scale function. In many cases they serve only to do calculation work or to furnish reference setpoint data to individual analog controllers. The role of the microprocessor can be to control those functions which are too small to justify the expense of a minicomputer or to aid a larger computer system by taking the place of an analog controller.

In comparing the performance of a microprocessor to that of an analog circuit in an industrial environment, there are several areas that should be included. The process of design, installation, and final adjustment of a control system often involves numerous changes as unforeseen problems occur. Changing the structure of an analog circuit involves the expense of material and labor necessary to rewire the circuit. Changing the structure of a microprocessor controller could involve rewiring when new peripheral devices are needed but will more often require only the alteration of the software programming. The accuracy of an analog controller is limited by such

parameters as operational amplifier drift, component tolerances and noise immunity. The resolution of a microprocessor controller is limited by the word size used, the method used for arithmetic operations and the capabilities of the peripheral devices. The major difference in performance of the two types of controllers is that the more complex methods of control incorporate on-line calculations that cannot be accomplished with analog circuitry. The microprocessor has the capability of performing more difficult mathematical functions, but the cycle time of the microprocessor used may require the addition of specialized digital circuitry in order to achieve the speed necessary to operate in a real-time situation.

## Future Trends

As digital technology advances, microprocessors will undoubtedly improve in speed and achieve greater versatility. They will become more independent of peripheral devices, such as external multiplication circuits, because they will have the necessary software speed for fast multiplication. This added speed will extend their usefulness in the fields of control and instrumentation. These concepts have already been partially realized by the recent development of bipolar LSI chips. Bipolar technology has previously been considered less dense than Metal-Oxide Semiconductor (MOS) and therefore not a prime

choice for LSI marketing. But the significant advantage of speed has kept integrated circuit researchers actively investigating concepts by which bipolar LSI might become more dense than MOS[10]. An example of this research activity is the announcement of a set of LSI chips containing Schottky bipolar sixteen-bit computing elements by Intel Corporation. This new concept is largely responsible for performance of up to two orders of magnitude faster than most p-channel MOS microprocessors[13].

The microprocessor employed for this investigation is the L-series MOS microprocessor made by Control Logic, Incorporated, which is structured around the INTEL 8008 CPU. Its internal functions will be briefly described in Chapter II which will also contain an analysis of the LSI central processor. Chapter III will detail the peripheral support designed to aid in processing a typical digital control algorithm. Chapter IV will explain the method of obtaining the PID control algorithm and describe the results of its implementation. Chapter V presents conclusions from the study and gives recommendations for future applications of microprocessors.

# CHAPTER II

## MICROPROCESSOR SYSTEM ORGANIZATION

The microprocessor selected for this project is the L-series microprocessor made by Control Logic, Incorporated. A general overview of the system architecture is presented in this chapter.

### Basic System Organization

The L-series microprocessor is an eight-bit, parallel machine that is structured around the INTEL 8008 LSI-MOS central processor. Its organization can be subdivided into three basic sections: (1) Central Processing Unit, (2) Memory, (3) Input-Output Control. These sections communicate with one another by control signals and transfer data by a series of busses as illustrated in Figure 2.1.

The microprocessor can be easily and conveniently programmed. The basic instruction set contains 48 instructions which cover over 180 combinations of register-register, register-memory, or input-output operations. A teletype interface provides program loading via a keyboard or a low-speed, paper-tape reader. Control Logic, Incorporated, has developed supporting software in a program called the Octal Debugging Technique (ODT). This program enables the user to load a program temporarily, alter memory locations within the program, and execute the program completely by teletype commands.

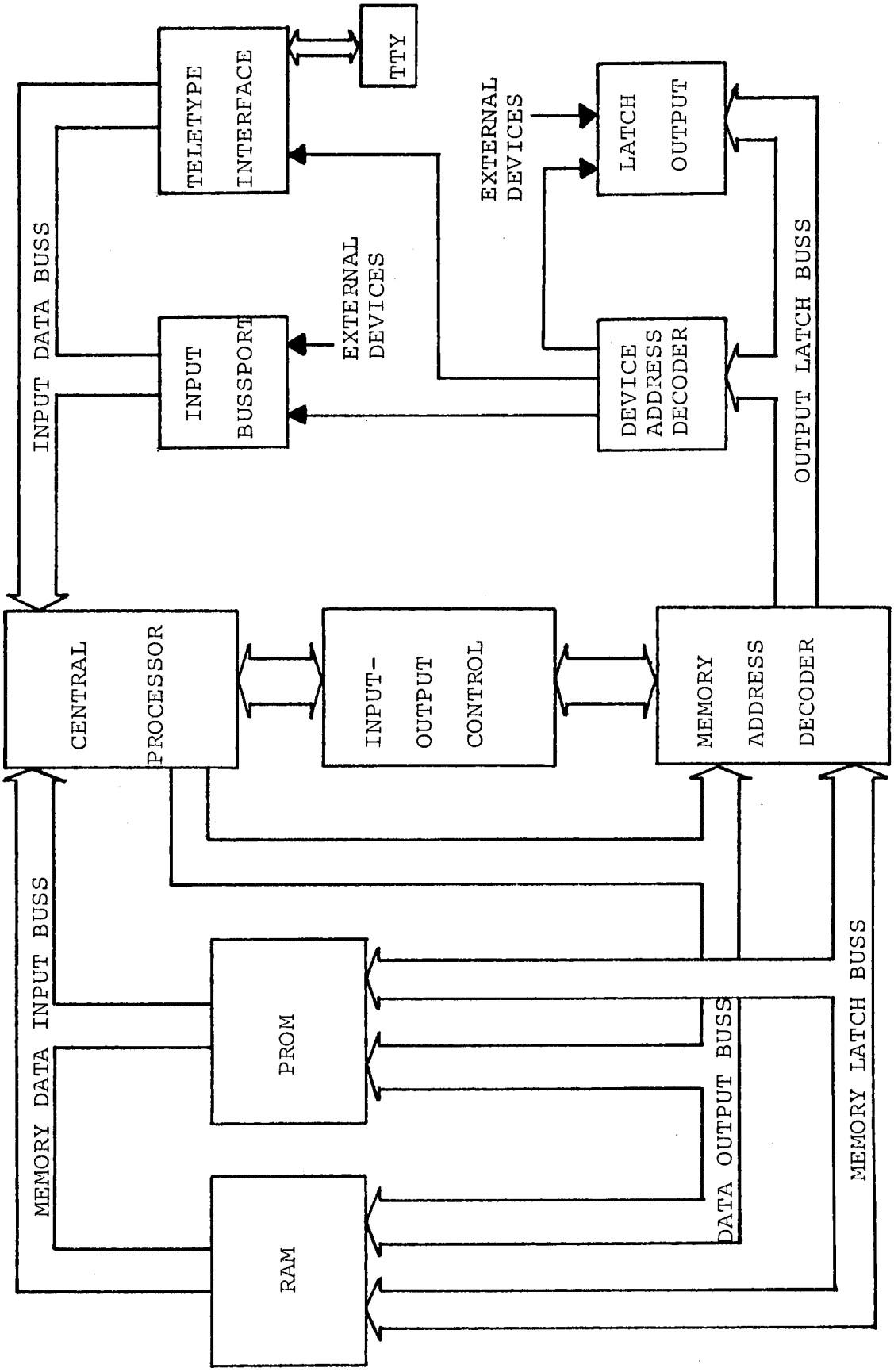



Figure 2.1. Microprocessor architecture.

The memory structure is composed of two types of memory. Random Access Memory (RAM) is a temporary area in which locations can be readily altered while in operation. Programmable Read Only Memory (PROM) is a semi-permanent area in which locations can be altered only by ultraviolet light exposure and reprogramming externally. In this particular machine, eight bits form one memory byte and one byte is the same as one word. The maximum memory capacity of 16,384 bytes is subdivided into four chapters of 4098 words each which in turn are subdivided into sixteen pages of 256 words each. This memory can consist of either RAM or PROM as determined by the user.

External control of the microprocessor is provided by console control and a priority interrupt structure. The console control allows the user to interrupt the central processor, execute an instruction, enter data, or step through a program a word at a time. The priority interrupt system handles requests for processor control from up to eight external devices with user-assigned priorities.

The system is implemented on 3 x 5 inch printed circuit boards with 44-pin edge connectors. The cards are contained in a frame which provides wire-wrap connections.

## Central Processing Unit

The main functions of the Central Processing Unit (CPU) are to fetch and decode instruction words and to perform logical and arithmetic operations based on the

decoded instructions. The main element in the CPU is the INTEL 8008 central processor. It contains an eight-bit, bidirectional data buss; two control inputs; four control outputs; six eight-bit data registers; two eight-bit temporary registers; an eight-bit accumulator; four condition flip-flops; an 8 x 14 bit push-down stack; and an eight-bit parallel binary arithmetic unit.

One of the two control inputs is a READY signal generated by the user via the console control. Its function is to halt the processor. The other control input is an INTERRUPT command generated by the Input-Output (I/O) section via the priority interrupt structure. It signals the processor to stop operation and service an external interrupt.

Instructions are executed by a sequence of eight possible processor states chosen by the decoded instruction. One instruction requires from one to three cycles through either three, four, or five processor states. A diagram illustrating all possible sequences of states is shown in Figure 2.2. States T1 and T2 load the low and high orders of a memory address. State T3 fetches the instruction or data from memory. States T4 and T5 execute the instruction. If the user has signaled the processor to stop via the console control, the processor will remain in the WAIT state after state T2. If a halt instruction is encountered, the processor enters the STOPPED state until interrupted by the user via the console control. Depending on the instruction, the cycle may end after state T3 or T4. Regardless of

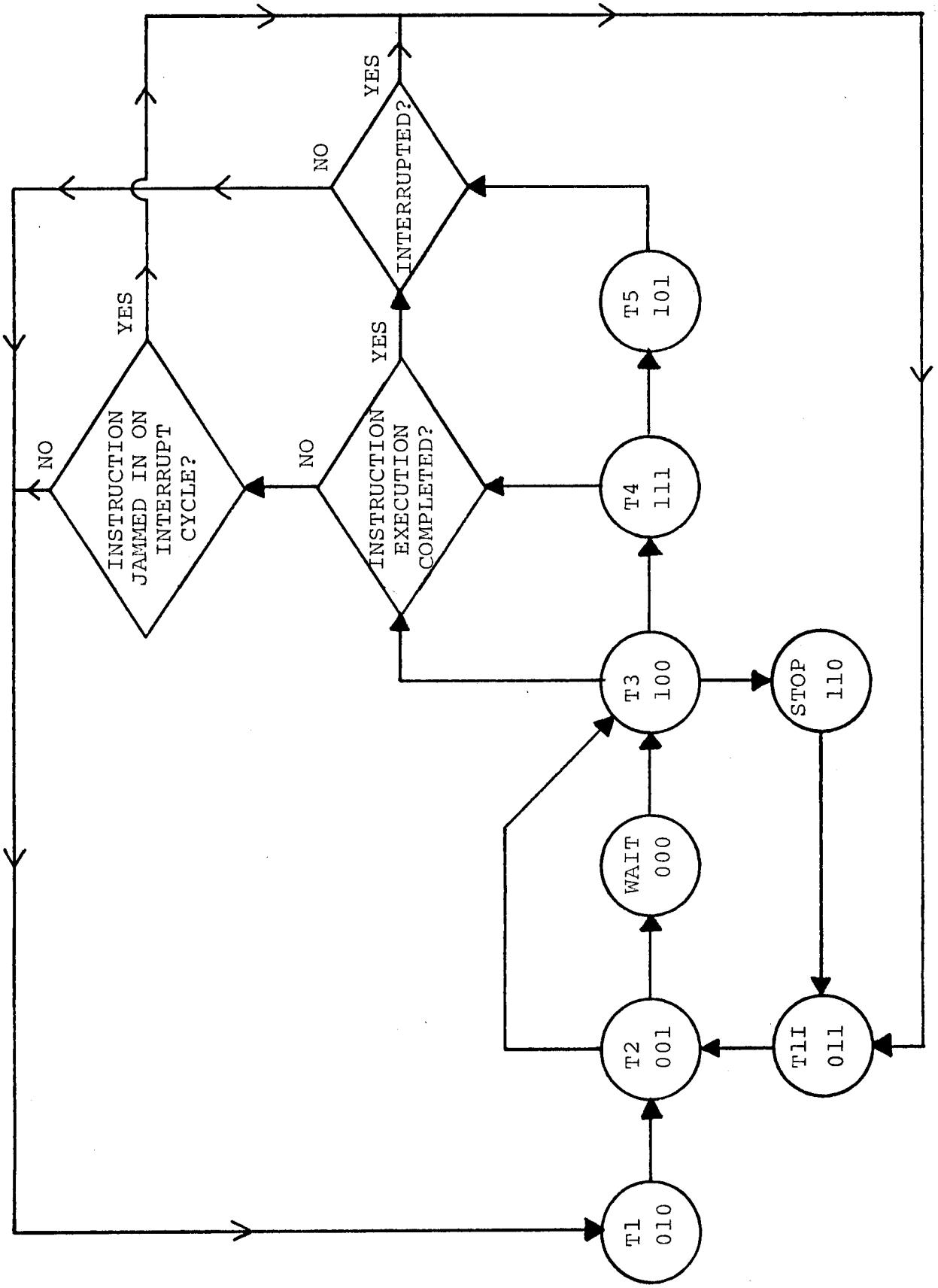



Figure 2.2. CPU state transition diagram.

where the cycle ended, the processor will enter state T1 unless an interrupt has occurred. Upon interrupt, the processor will substitute state T1I for T1 which effectively stores the program counter in the pushdown stack and jumps to a location designated by the priority interrupt structure. The central processor codes its present state into three bits (S0, S1, S2) which are three of the control outputs. The fourth control output is a synchronizing pulse (SYNC) that coordinates external devices to the central processor. Figure 2.3 illustrates the architecture of the CPU.

The eight-bit parallel binary arithmetic unit performs logical operations, comparison, addition and subtraction. It also sets the Carry, Zero, Sign and Parity condition flip-flops as a result of those operations. They are tested by conditional jumps, subroutine calls and subroutine returns. These subroutine operations employ the pushdown stack which operates on a first-in-last-out principle. Its eight levels provide nesting of up to seven subroutines.

The central processor is supported by a data buss multiplexer and a state decoder to complete the functions of the CPU. The data buss multiplexer, enabled and controlled by the Input-Output Section, chooses which of the data busses to connect to the central processor's bidirectional data buss. The state decoder transfers the three-bit state

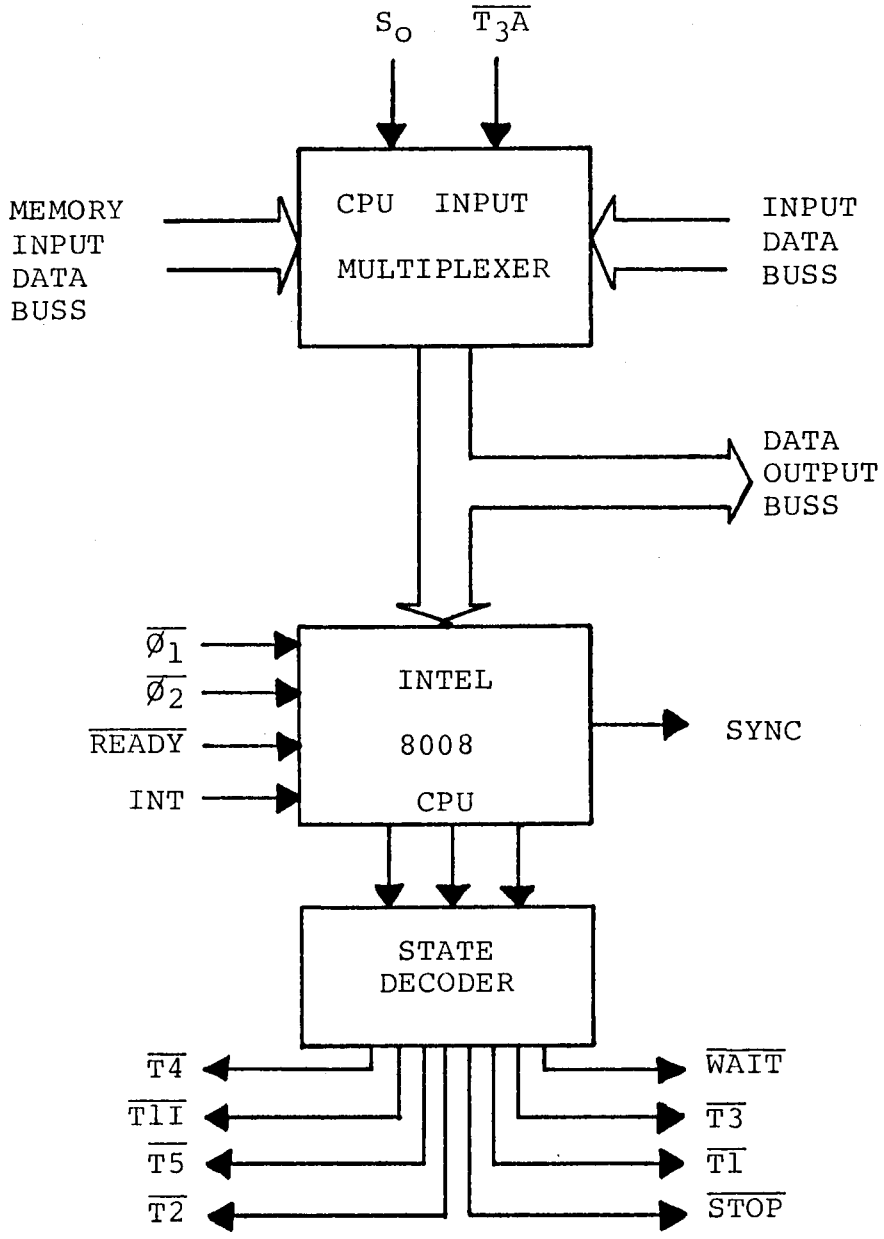



Figure 2.3. CPU architecture.

code into eight control signals associated with the eight processor states.

Input-Output Structure

The Input-Output (I/O) section of the microprocessor, containing all of the timing controls as well as peripheral communications interfacing, is the most complex. It provides the two-system clock signals ($\overline{\emptyset 1}$ and $\overline{\emptyset 2}$) which cycle the central processor.

The I/O section consists of the I/O Driver, Device Address Decoder, Priority Interrupt System, Buss Port and Latch Output. The I/O Driver coordinates the control signals from the CPU, Memory and Priority Interrupt System. Figure 2.4 illustrates the timing of the control signals in the I/O section. At state times T1 and T2, the I/O Driver provides control signals (LLA and LHA) to the Memory section to load the low and high order memory address. The two highest order bits in this address are decoded to distinguish between a memory write, memory read, or an input-output instruction. It is at this point that state T3 is anticipated and control signals ($\overline{T3A}$ and SO) are sent to the CPU data buss multiplexer.

The Priority Interrupt System allows the user to divide interrupts into an Immediate Group and a Deferred Group. When an Immediate Interrupt is present, the microprocessor immediately processes the request. When a Deferred Interrupt is present, the microprocessor will wait until it encounters a user-supplied output instruction

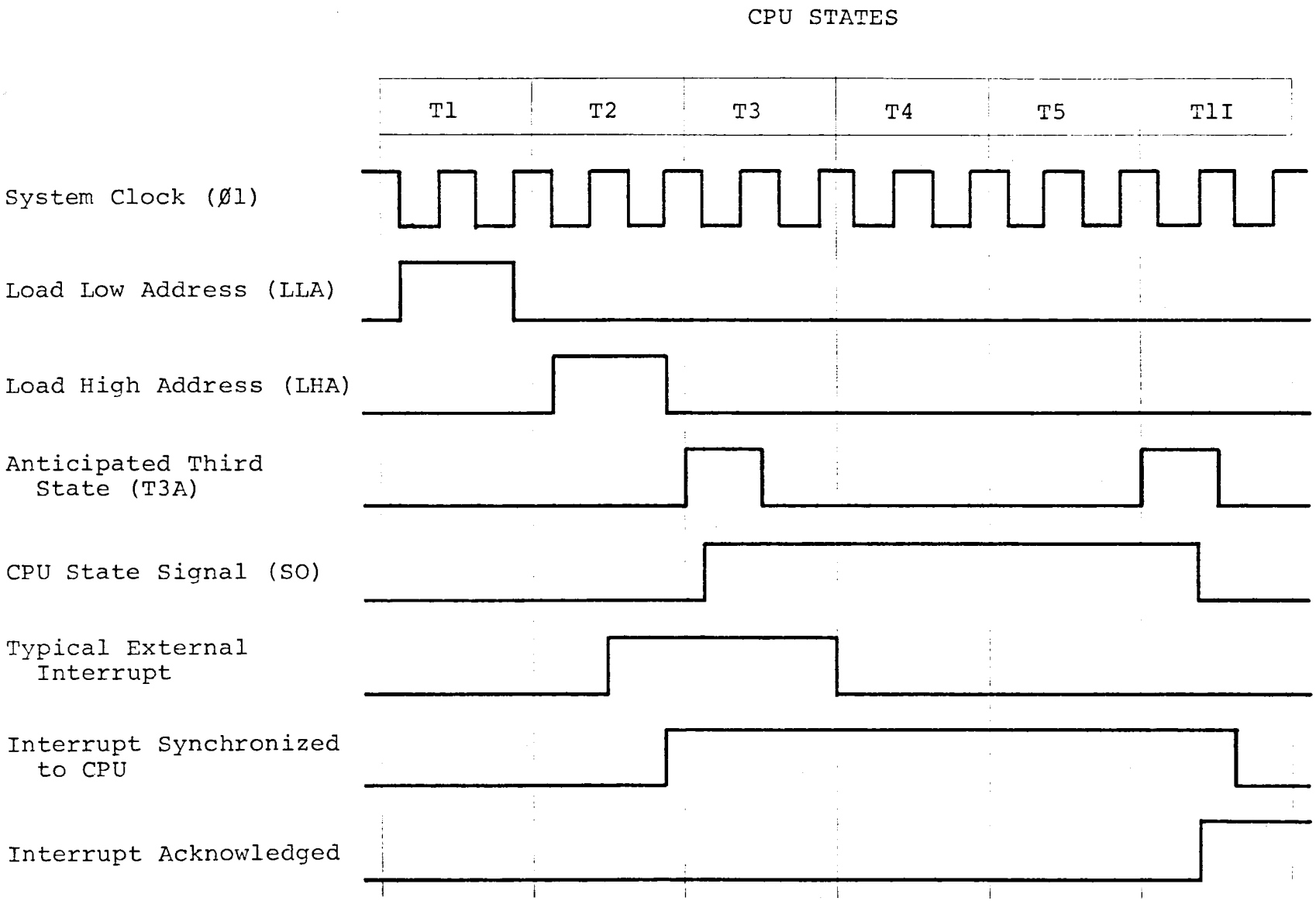



Figure 2.4. I/O signal timing diagram.

before processing the request. Interrupts are processed according to user-assigned priority with the Immediate Group having the higher priority. The interrupt is processed by loading a RESTART instruction during state T1I. The destination of the restart is a particular location in PAGE 0 memory where the user may program a subroutine.

The Device Address Decoder has the capability of decoding a four-bit address into sixteen control signals. Its purpose is to signal devices during I/O procedures. The I/O Driver generates the control signals ($\overline{IN}$ and $\overline{OUT}$) to enable the decoder.

The Buss Port and the Latch Output provide respective input and output links with data busses when enabled by the Device Address Decoder. They provide a path of communication with external peripherals.

## Memory Organization

Memory for the L-series microprocessor is available in three packages, each with its own memory address decoding logic. RAM can be obtained in either 256 bytes per card or 1024 bytes per card. PROM is available in only 512 bytes per card.

The RAM is made up of eight integrated circuit chips with one bit of the memory byte stored in each chip. The 256 byte RAM consists of 256 bit chips, and the 1024 byte RAM consists of 1024 bit chips. This type of RAM is static, meaning that the memory states are lost when the

power is lost. The microprocessor in this particular project has 1024 bytes of RAM and 512 bytes of PROM.

The PROM will retain its memory state with the power off. The only method of altering the memory is to completely erase the PROM by ultraviolet light exposure and then reprogram the entire PROM. Erasure is accomplished by exposing the PROM through the small quartz window on the top of the package to high intensity shortwave ultraviolet light at a wavelength of 2537 Angstroms. Exposure for two to three minutes will adequately erase the PROM. The PROM is programmed via the Programmer Card available from Control Logic. This card supplies the control signals and voltages necessary to program a PROM.

All memory is addressed by the Memory Address Storage (MAS) module. This module directly addresses up to 16,384 bytes of memory and can be wired to indirectly address up to 65,536 bytes. The MAS receives the memory address in communicating with the I/O Control and sends a 14-bit address field to the memory blocks. The communication of the I/O and MAS has been explained in Section 3 of this chapter and is further illustrated by Figure 2.5. This address field is subdivided into three fields of eight bits, four bits, and two bits. The eight-bit field consists of the least significant eight bits, and the two-bit field consists of the most significant two bits. The 512-byte PROM memory block decodes the eight-bit field to select one of 256 bytes of a PROM package and decodes the four-bit field to select

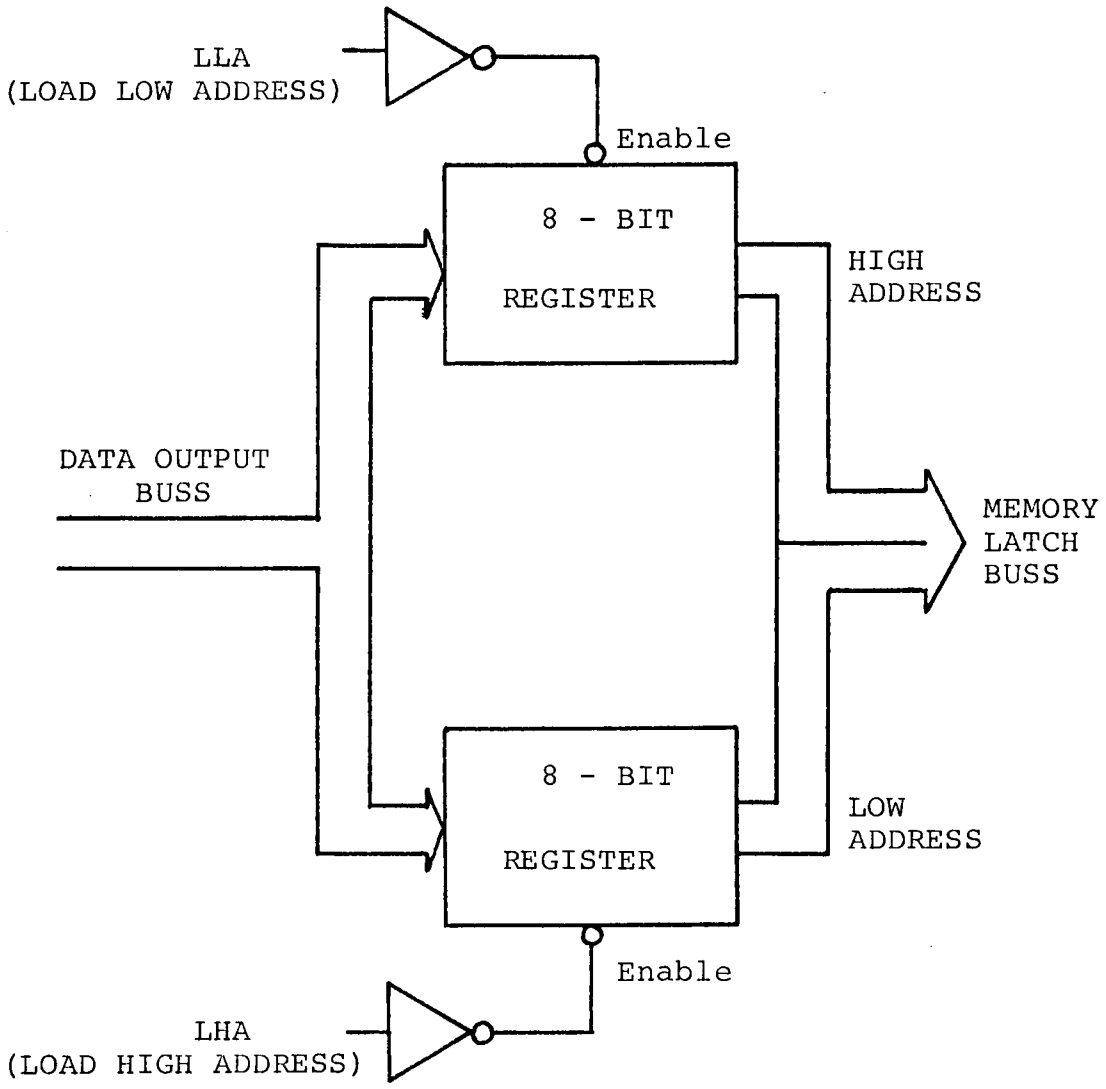



Figure 2.5. Memory address storage.

one of 16 pages. The two-page selections are determined by the user via jumpers on each PROM memory card. The 256-byte RAM block decodes the eight-bit field to select one bit from each of the eight integrated circuits to form one byte. The four-bit field is decoded as on the PROM card except that only one of the 16-page select lines is used. The 1024-byte RAM block uses the eight-bit field and the two least significant bits in the four-bit field to select one of 1024 bits on each of the eight integrated circuits. The remaining two bits in the four-bit field are used to select one of four chapters of four pages each. Therefore, one 1024-byte RAM card can only be used for four consecutive pages beginning with pages 0, 4, 8, and 12. The choice of the chapter is user-selectable via jumpers. In all memory blocks, the two-bit field is used to enable the four-bit decoder, thus dividing 16,384 bytes of memory into four chapters of 16 pages each.

The MAS section contains one PROM package of 256 bytes that is pre-wired to page 0. This gives the user the convenience of having a specialized PROM always included in the system where interrupt subroutines can be programmed.

A basic description of the L-series microprocessor has been presented. The following chapter continues in the development of the microprocessor control system as the design and justification of peripheral digital circuitry is presented.

# CHAPTER III

## PERIPHERAL DESIGN

The control application of this microprocessor necessitates the aid of peripheral devices in order to effectively perform its function. The design philosophy, as well as the justification for these peripherals, is detailed in the following sections.

### Signal Servicing

The block diagram in Figure 3.1 illustrates the basic system configuration. Two analog signals, feedback and error, must be interfaced to the microprocessor. The feedback signal is sampled and converted by the use of a DATEL SHM-2 Sample-Hold Amplifier (SHA) and a DATEL ADC-EH1 Analog-to-Digital Converter (ADC). The error signal is produced by the numerical calculation of the microprocessor and applied to the Plant System by means of a DATEL DAC HB10B Digital-to-Analog Converter (DAC).

The justification of the ADC and DAC is intuitively obvious for they are standard interfaces between the digital and analog worlds, but the use of the SHA must be reasoned. As shown in Figure 3.2, the ADC operates by cycling a digital reference through a series of bit patterns which are converted to an analog voltage by means of a very fast internal DAC. The DAC voltage is summed negatively with the analog input voltage, and the resulting voltage is

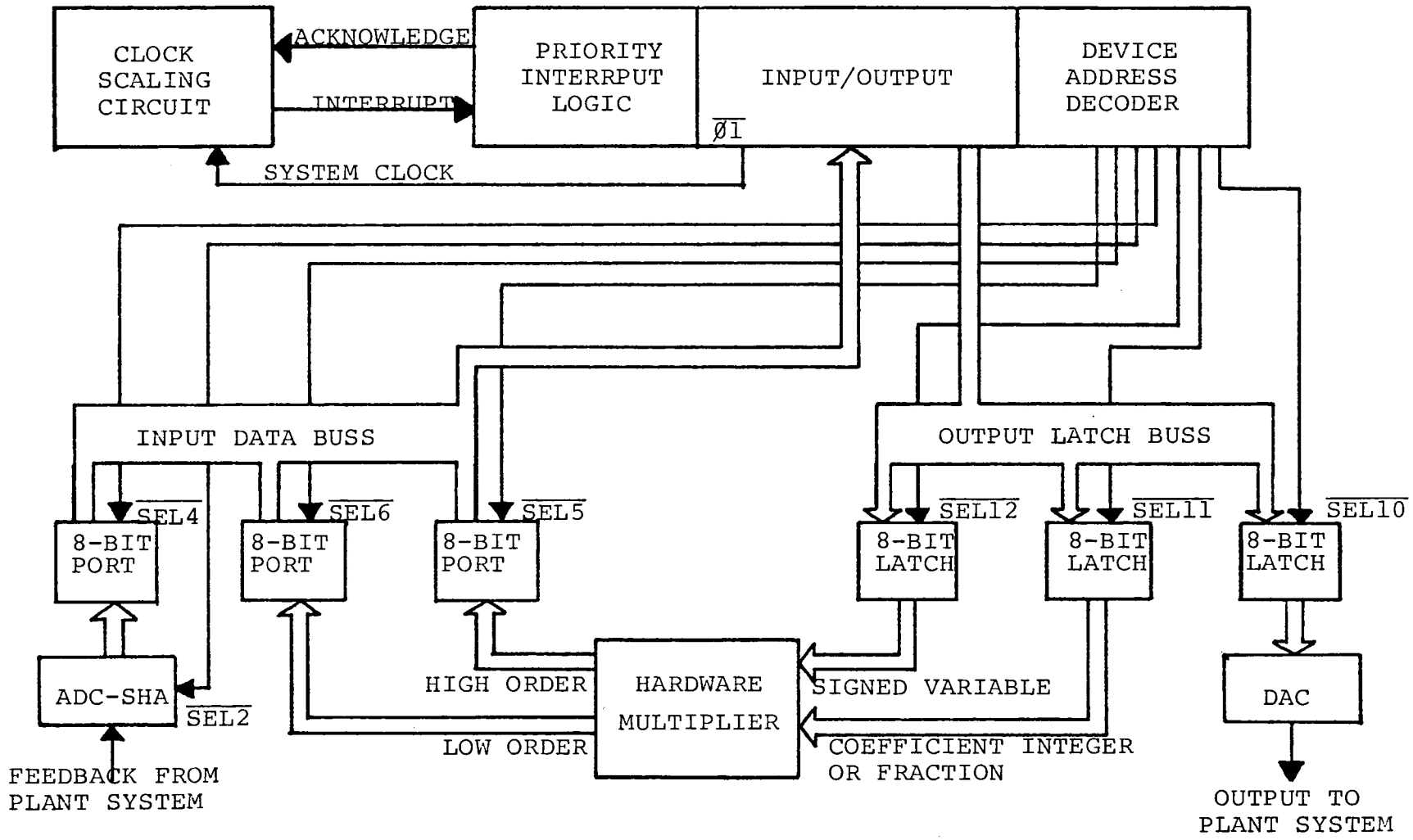



Figure 3.1. Peripheral interfacing structure.

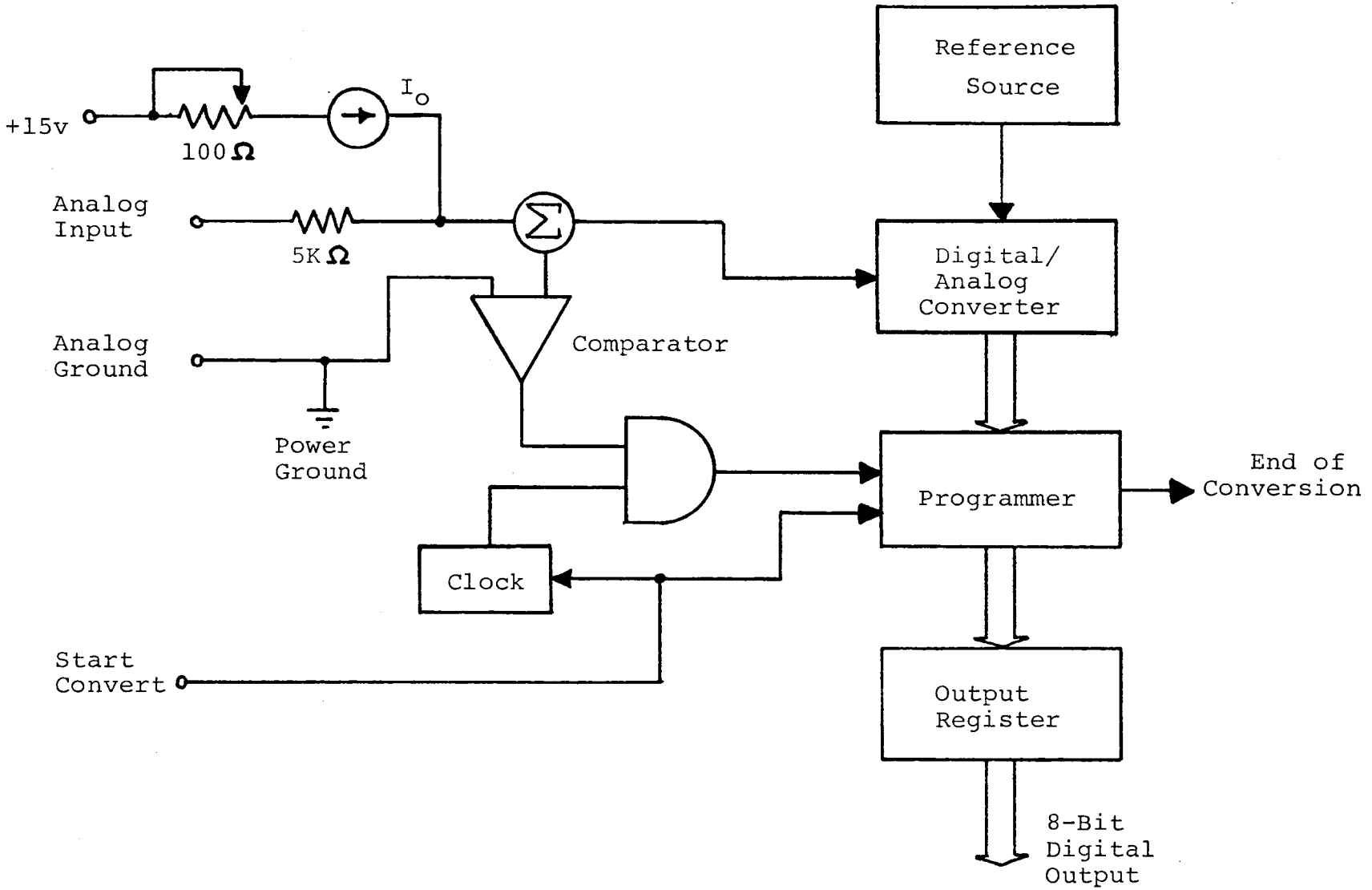



Figure 3.2. ADC architecture.

compared to ground via an analog comparator with a logical output. When the two voltages are equal within certain limits, the logical output of the comparator signals the programmer to output that bit pattern to an output register. The worst case conversion time is four microseconds. If in this time span the analog input voltage changes by an amount exceeding the limits of the comparator, the ADC process can result in a false conversion or no conversion at all. The SHA insures a constant input voltage for the ADC to operate on at a delay of only one microsecond.

The DAC requires only data line interfacing with the Latch Output of the microprocessor. The ADC-SHA combination, however, requires a control line interface as well as a data line interface. The interfacing is shown in Figure 3.3. The device select lines are a result of the decoding of input instructions. These instructions require six processor states, each for a total analog signal input time of 48 microseconds. Twenty-four microseconds are used for the actual analog-to-digital conversion which yields significant tolerance to the five microsecond conversion rate of the SHA-ADC combination.

## Clock Scaler

Classical digital control theory requires that sampling be done at a consistent interval. The most convenient method for consistent sampling is by interfacing the Priority Interrupt Structure with a clock pulse. The

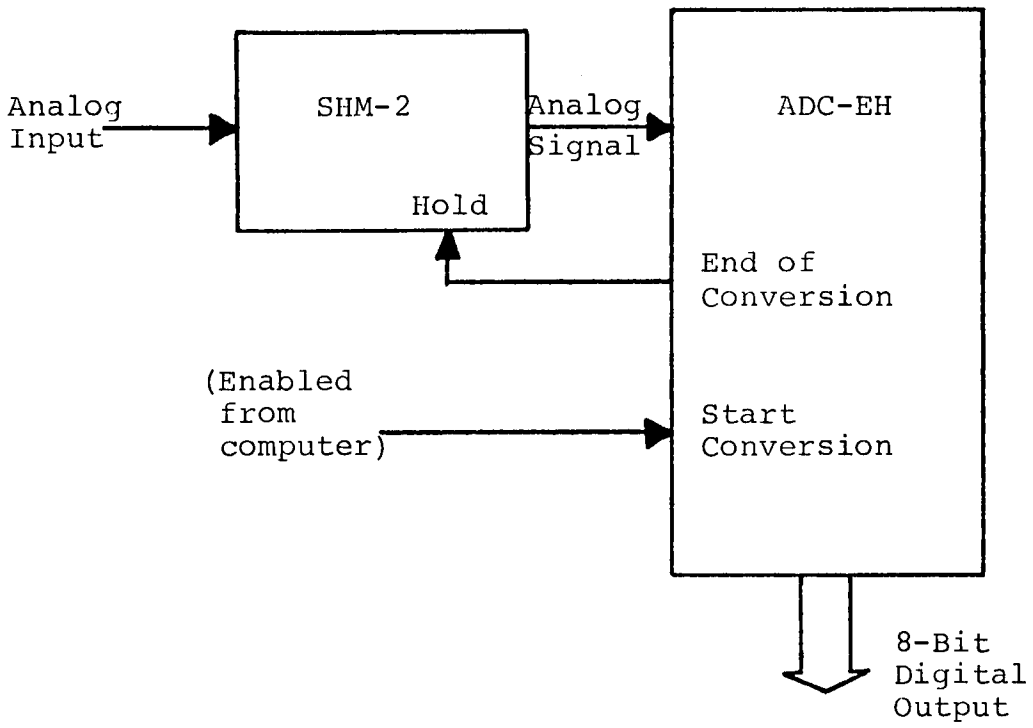


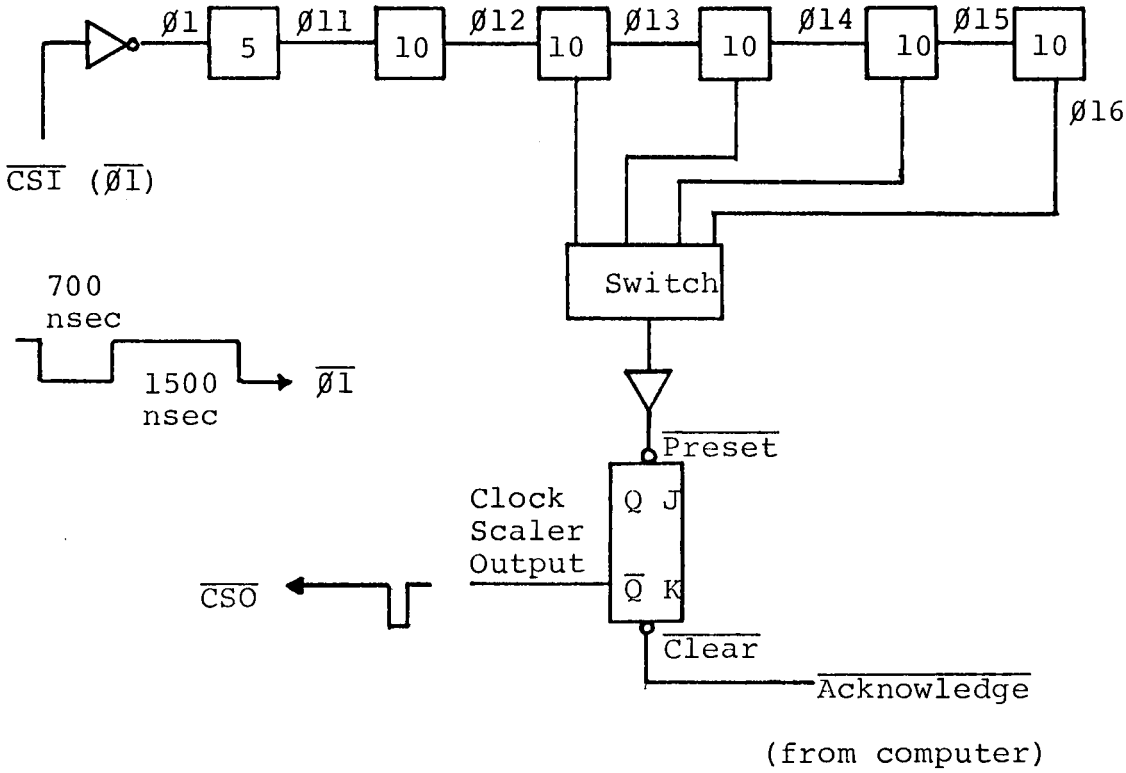

Figure 3.3. Interconnection of SHA to ADC.

microprocessor is driven by two pairs of monostable multivibrators, both of which are readily available for external application. The clock $\overline{\emptyset 1}$ operates at a 500 KHz frequency which lends itself well to scaling circuitry as shown in Figure 3.4. The theory behind a scaler is to change output levels for every N number of input pulses, thereby dividing the input frequency by N. Asynchronously coupling scalers, as shown in Figure 3.4, produce selectable sampling frequencies in decade multiples. The scaling circuitry is interfaced with the microprocessor through a J-K flip-flop. The output of the flip-flop ($\overline{CSO}$) is set by the clock scaler and is connected to the microprocessor's Priority Interrupt structure. After receiving the interrupt, the Priority Interrupt structure resets the flip-flop by means of an acknowledgment signal ($\overline{ACK}$).

Two different types of scalers are needed. Even though their scaling factors are different, they basically operate in the same fashion, as shown in Figures 3.5 and 3.6. They consist of J-K flip-flops connected together to form a counter which sequences as shown also in Figures 3.5 and 3.6. A unique bit pattern is then detected by the NAND gate, and the output of the NAND gate provides a negative-going pulse signal.

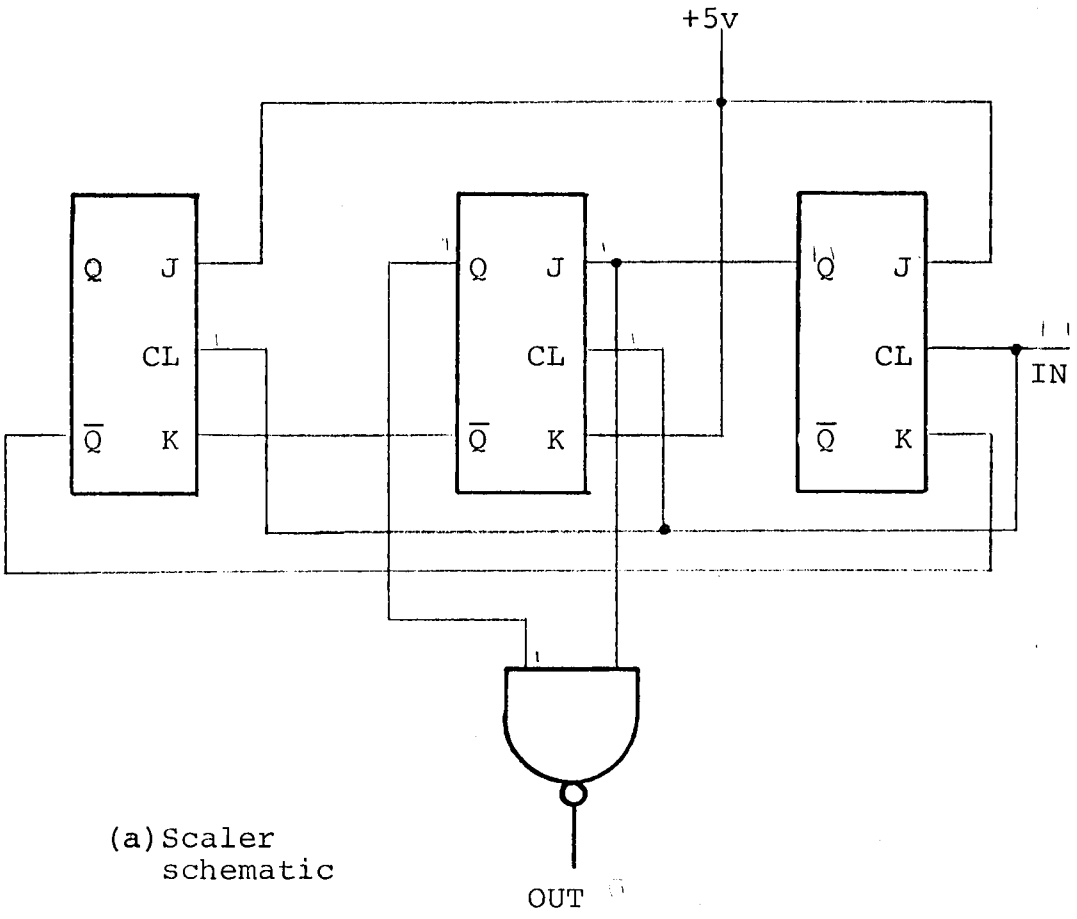
Multiplier

The algorithm chosen to be programmed in the microprocessor is of the form

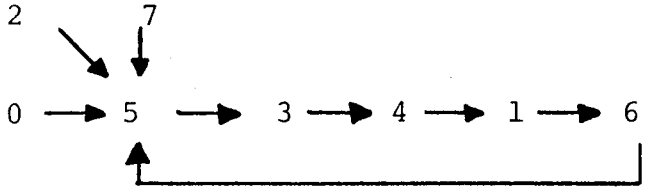



| | Period | Frequency |
|---|---|---|
| Ø1 | 2 μsec | 500 khz |
| Ø11 | 10 μsec | 100 khz |
| Ø12 | 100 μsec | 10 khz |
| Ø13 | 1 msec | 1 khz |
| Ø14 | 10 msec | 100 hz |
| Ø15 | 100 msec | 10 hz |
| Ø16 | 1 sec | 1 hz |

Figure 3.4. Basic configuration of clock scaler.

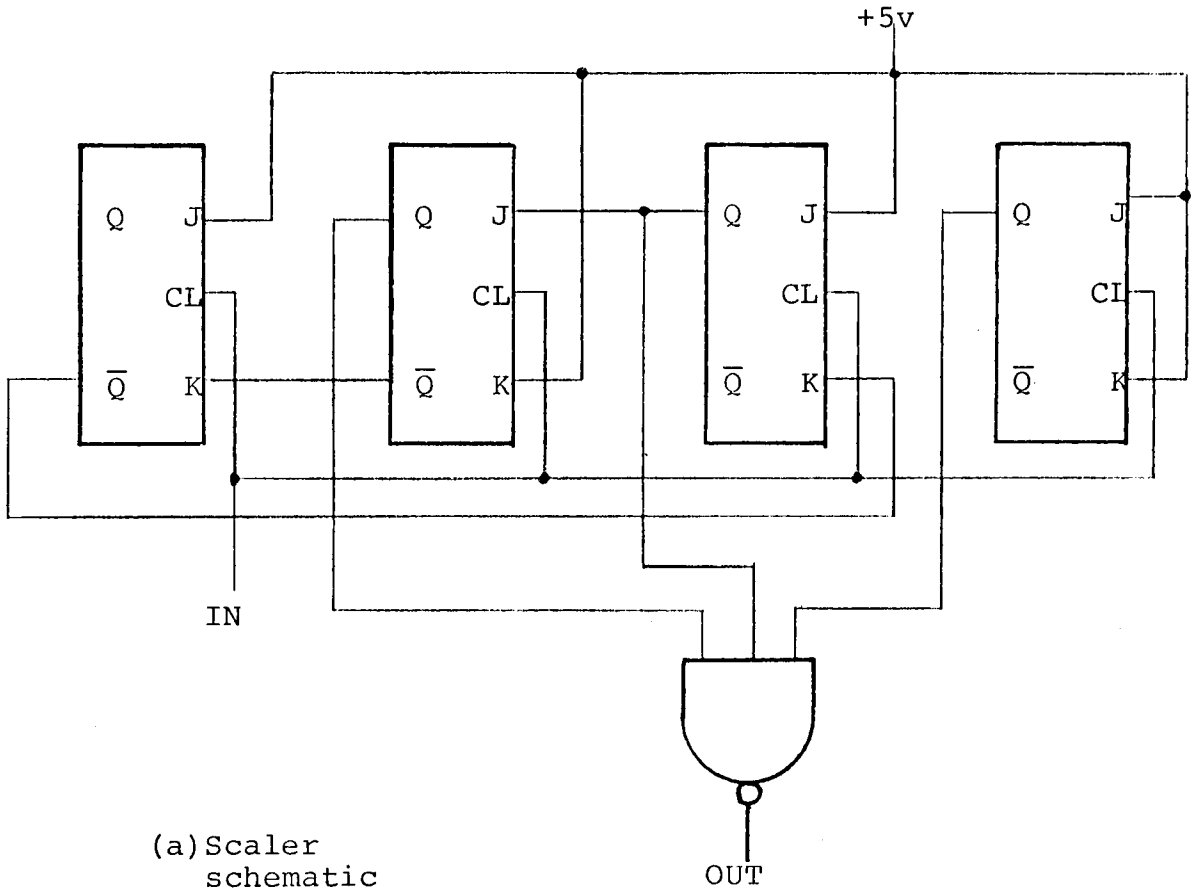



(a) Scaler schematic

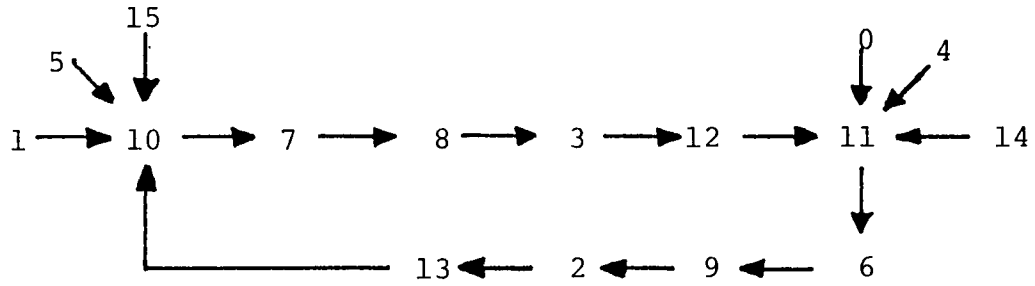



(b) Scaler cycle pattern

Figure 3.5. Frequency scaler with factor of five.




(a) Scaler schematic




(b) Scaler cycle pattern

Figure 3.6. Frequency scaler with factor of ten.

$$S = A*E + B*E1 + C*E2 + D*S1 + F*S2$$

where A, B, C, D and F are coefficients; E, E1 and E2 are respectively present, first past and second past values of the error signal; and S, S1 and S2 are respectively present, first past and second past values of the algorithm output. The use of this form will be discussed in Chapter IV.

By virtue of the fact that there are multiplications contained in the algorithm, it is necessary to evaluate methods of multiplication. The fastest software multiplier of two eight-bit numbers found for this microprocessor is shown in Figure 3.7. The total amount of time necessary to process this software is 3.008 milliseconds. Complicating the matter somewhat is the fact that the coefficients A, B, C, D and F are sixteen-bit entities with one eight-bit group representing an integer part and the other eight-bit group representing a fractional part. Therefore, multiplying this type of number by an eight-bit signed integer variable requires adding the higher order eight bits of the integer-variable product to the sixteen bits of the fraction-variable product. This method truncates the fractional part of the ultimate product, losing a small amount of resolution. This method also requires two multiplications to achieve the result. Another problem is the matter of the signs of the two numbers, since the software multiply deals only with unsigned integers. Software will be needed to convert negative numbers to positive numbers and vice versa. The

| Label | Mneumonic | No. States | Comment |
|---|---|---|---|
| MPY | XRA | 5 | Clear all summing |
| | LBA | 5 | registers. |
| | LCA | 5 | |
| | LEA | 5 | |
| | LLI 010 | 8 | Set index to $10_8$. |
| MULTO | DCL | 5 | Decrement index, and |
| | RTC | 5 | Return to call if zero. |
| | LAD | 5 | Rotate least significant |
| | RAR | 5 | multiplier bit into |
| | LDA | 5 | carry, and restore. |
| | JFC MULT1 | 9 | Jump if carry bit is zero. |
| | LAC | 5 | Add low order shifted |
| | ADH | 5 | multiplier to low order |
| | LCA | 5 | product sum, and restore. |
| | LAB | 5 | Add high order shifted |
| | ACE | 5 | multiplier to product |
| | LBA | 5 | sum with carry, and restore. |
| MULT1 | LAH | 5 | Shift multiplier in E |
| | RAL | 5 | left one bit through H |
| | LHA | 5 | via carry. |
| | LAE | 5 | |
| | RAL | 5 | |
| | LEA | 5 | |
| | JMP MULT0 | 8 | Go to MULT0 |

WORST CASE EXECUTION TIME

Set up for multiplication loop: (28 states)*(4 microseconds/state) = 112 microseconds

Execute 7 loops: (102 states/loop)*(7 loops)* (4 microseconds/state) = 2856 microseconds

Exit: (10 states)*(4 microseconds/state) = 40 microseconds

TOTAL EXECUTION TIME = 3.008 milliseconds

Figure 3.7. Program structure and timing characteristics of a software multiply subroutine.

software multiply also requires the use of all of the general purpose registers. This means that all intermediate results must be stored in memory. Figure 3.8 shows the programming flow chart and timing that is necessary to process the multiplication of one coefficient by one variable. The total time used in processing a five coefficient algorithm is 40.076 milliseconds.

For many digital control techniques, computation time is a critical factor. It is well known in digital control theory that insufficiently slow sampling rates can cause instability. A generally accepted rule is that computations should be accomplished at one-tenth of the sampling rate to cause no ill-effects due to real-time delays. With optimal programming, the delay time can be reduced to 7.328 milliseconds with the total algorithm still taking 40.076 milliseconds. Figure 3.8 gives a summary of the individual execution times used to arrive at these time estimates. This timing is acceptable for sampling rates of 100 milliseconds and 1 second. However, it is possible for these rates to cause instability in some systems. Therefore, to make the system faster and more conservative with core, a hardware multiplication circuit is justifiable at an additional cost of less than $100.00.

From consideration for optimal programming, the coefficients are stored in positive sign only and the products are added or subtracted according to the coefficient's

ONE COEFFICIENT-VARIABLE MULTIPLY

| FUNCTION | EXECUTION TIME |
|---|---|
| Retrieve variable | .064 msec |
| Calculate magnitude and sign | .244 msec |
| Load in register | .020 msec |
| Retrieve integer | .064 msec |
| Multiply | 3.008 msec |
| Store result | .128 msec |
| Retrieve fraction | .064 msec |
| Multiply | 3.008 msec |
| Retrieve integer product | .128 msec |
| Add products | .140 msec |
| Adjust sign and magnitude | .460 msec |
| TOTAL EXECUTION TIME | 7.328 msec |

TOTAL ALGORITHM CALCULATION

| FUNCTION | EXECUTION TIME |
|---|---|
| Input A/D and calculate error | .470 msec |
| Update E1,E2,S1, and S2 | .476 msec |
| Multiply B,C,D,F, by E1,E2,S1,S2 | 29.312 msec |
| Sum first four products | .640 msec |
| Set up for A*E | 1.320 msec |
| Multiply A*E | 7.328 msec |
| Sum S | .140 msec |
| Output S to D/A | .390 msec |
| TOTAL EXECUTION TIME | 40.076 msec |

Figure 3.8. Application of the software multiply subroutine to a digital algorithm.

original sign. With this in mind, the hardware multiplier is designed to multiply an eight-bit signed integer by an eight-bit unsigned integer. The principle behind the design is that of a "shift-and-conditionally-add" technique. To demonstrate this technique, the multiplication can be represented by

$$(A_{128} + A_{64} + A_{32} + A_{16} + A_8 + A_4 + A_2 + A_1) * (B_{64} + B_{32} + B_{16} + B_8 + B_4 + B_2 + B_1) \quad (3.2)$$

where the A's and B's are binary bits and the subscripts denote the weighting factor associated with the bit position in the word. "A" represents the unsigned eight-bit integer and "B" represents the signed eight-bit integer. Only the magnitude of "B" is used in the actual multiplication which is why "B" has only 7 bits. Regrouping of 3.2 yields:

$$\begin{aligned} & B_1(A_{128} + A_{64} + A_{32} + A_{16} + A_8 + A_4 + A_2 + A_1) \\ & + B_2(A_{128} + A_{64} + A_{32} + A_{16} + A_8 + A_4 + A_2 + A_1) \\ & + B_4(A_{128} + A_{64} + A_{32} + A_{16} + A_8 + A_4 + A_2 + A_1) \\ & + B_8(A_{128} + A_{64} + A_{32} + A_{16} + A_8 + A_4 + A_2 + A_1) \\ & + B_{16}(A_{128} + A_{64} + A_{32} + A_{16} + A_8 + A_4 + A_2 + A_1) \\ & + B_{24}(A_{128} + A_{64} + A_{32} + A_{16} + A_8 + A_4 + A_2 + A_1) \\ & + B_{64}(A_{128} + A_{64} + A_{32} + A_{16} + A_8 + A_4 + A_2 + A_1) \end{aligned} \quad (3.3)$$

Since $B_n$ is a bit with a weight of n associated with it, multiplication by $B_n$ is either zero or A shifted left $(\log_2 n)$-1 number of bits. This is to say that if $B_n$ is one, do not add it to the product summation; and if $B_n$ is one, add the n-1 shifted result to the others. In hardware form,

the add decision is achieved with AND gates, and the shifting and adding is accomplished with ADDER gates. Figure 3.9 shows the hardware configuration of the multiplier using the block representations. The blocks labeled $ASA_i$ (i = 1 to 7) are shift-and-conditionally-add iterative cells which are shown in detail in Figure 3.10. The blocks labeled 8-bit COMPLEMENT are two's complement cells which are shown in detail in Figure 3.11.

The development of the digital control algorithm will be presented in the next chapter. An evaluation of the performance of the microprocessor control system as applied to an analog-simulated plant system will also be performed.

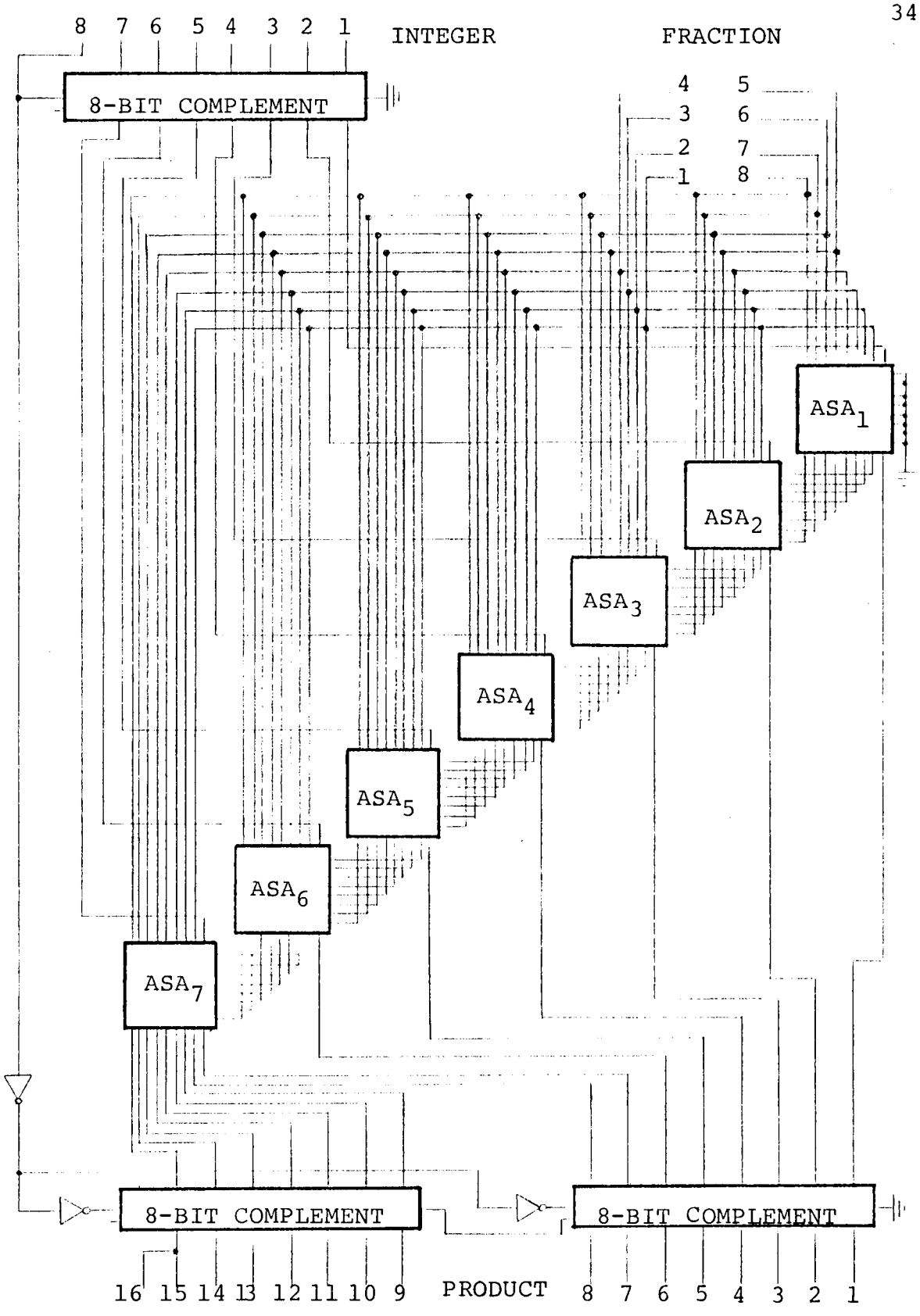



Figure 3.9. Iterative cell structure of hardware multiplier.

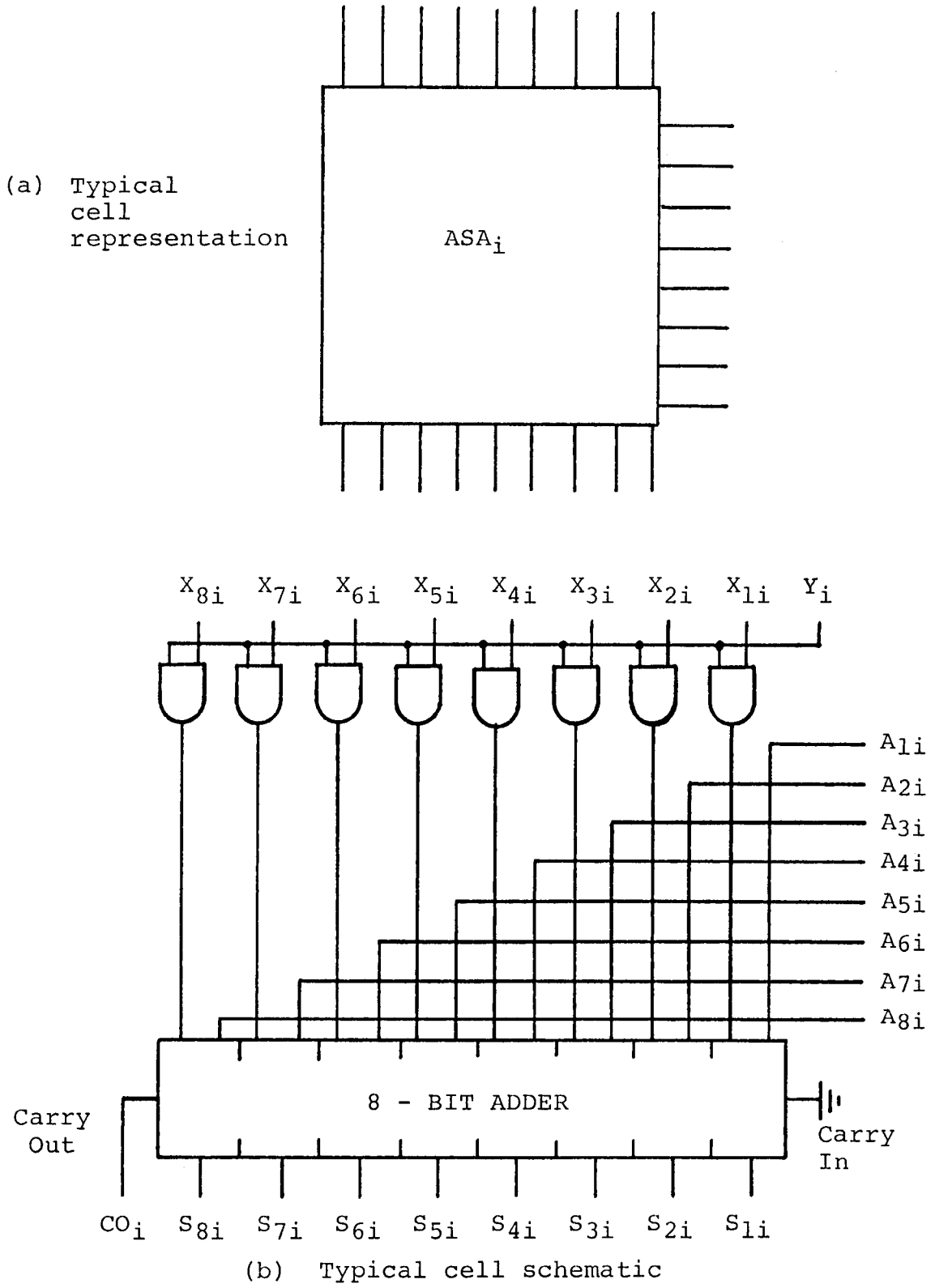
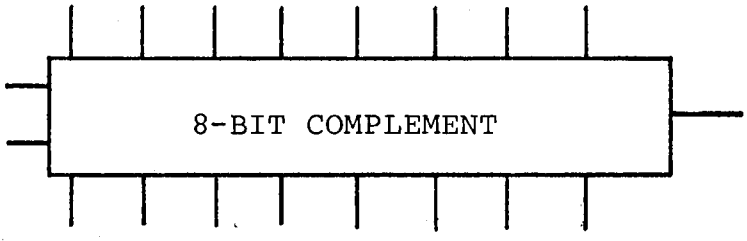
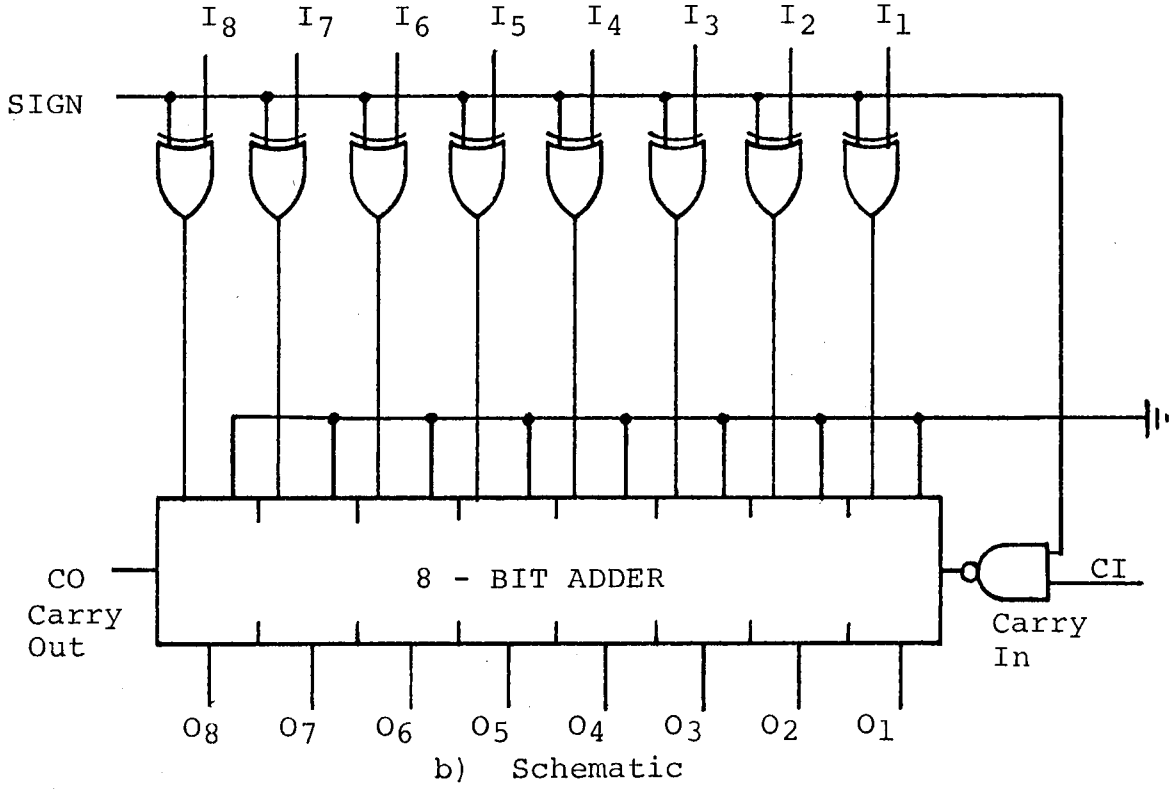



Figure 3.10. Shift-and-conditionally-add iterative cell.




a) Typical representation




b) Schematic

Figure 3.11. Eight-bit complement circuitry.

# CHAPTER IV

## DEVELOPMENT AND PROGRAMMING OF CONTROL ALGORITHM

The basic requirement of the algorithm structure is to demonstrate the microprocessor's capabilities and limitations in the field of real-time digital control while remaining as versatile as possible. This chapter explains the reasoning behind the selection of the PID control method as an example of digital control by a microprocessor. The method by which the digital algorithm form is derived and programmed is detailed, and in conclusion, the results of applying the microprocessor PID controller to an analog-simulated system are evaluated.

### PID Algorithm Background and Development

The analog PID controller has been a generally accepted and widely used means of control in the industrial environment for many years. The popularity of the PID can be attributed to several factors. It is a versatile controller in that it can be adjusted easily to achieve a desired response for many different systems. Two other important features are that it is relatively inexpensive and that it is a standard off-the-shelf item.

The analog PID can best be represented by three parallel branches summed together to form the output of the controller. The branches contain the input, the integral of the input with respect to time, and the derivative of

the input with respect to time. Each branch has an adjustable gain which allows the user to tune the controller by summing portions of the signal in each branch. A typical representation of the analog PID is shown in Figure 4.1.

The popularity and success of the analog PID has encouraged a number of engineers to develop and implement a digital form of the controller. The digital PID, however, is not always economically justified over the analog PID. When implementing a large number of controllers, it is possible to justify the expense, but this number of controllers is generally larger than most typical processes need. An application in which a digital system might well be justified is one where either the limited output range or gain settings of a common PID controller are not sufficient to accurately control the process.

The analog PID controller can be represented mathematically by:

$$m(t) = K_c \left[ e(t) + \frac{1}{T_i} \int_0^t e(t)dt + T_d \left(\frac{de(t)}{dt}\right) \right] + m_r \qquad (4.1)$$

where

$K_c$ = proportional gain

$T_i$ = reset or integral time

$T_d$ = derivative time

$m_r$ = initial system condition

$m(t)$ = output of the controller

$e(t)$ = input to the controller

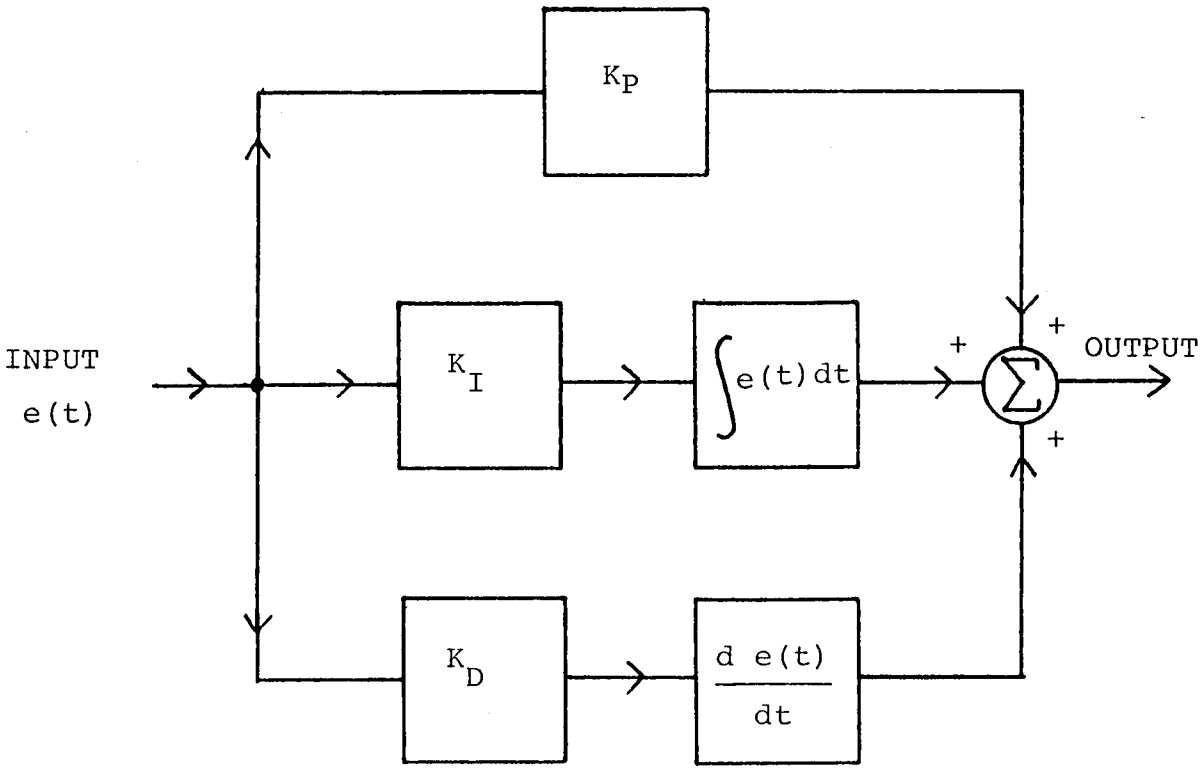



Figure 4.1. PID block diagram representation.

There are several ways of digitally processing this algorithm, of which one of the more common is to simulate an approximation of the integral and derivative of the input with respect to time. The integral can be approximated by summing the products of the input value e(t) and sampling time increment. This method effectively yields an approximate area under the two-dimensional curve of the input signal versus time. The derivative is simulated by calculating the slope of the input signal versus time curve at the time of the sampling. This calculation is accomplished by subtracting the last sampled input value from the present sampled input value, and then dividing that result by the sampling time increment(5).

Smith (5) suggests using a more direct algorithm.

$$m_n = K_c\left[e_n + \frac{T}{T_i}\sum_{k=0}^{n} e_k + \frac{T_d}{T}(e_n - e_{n-1})\right] + m_r \qquad (4.2)$$

$m_n$ = value of algorithm output at the nth sample

$e_n$ = value of the algorithm input at the nth sample

$T$ = sampling period

This representation of the digital PID algorithm is commonly referred to as the position form. Rewriting Equation 4.2 in terms of one sample period delay ($m_{n-1}$) and subtracting the result from the original equation yields

$$m_n - m_{n-1} = K_c(e_n - e_{n-1}) + \frac{K_c T}{T_i}e_n + \frac{T_d K_c}{T}(e_n - 2e_{n-1} + e_{n-2})$$

Equation 4.3 is commonly referred to as the velocity form of the algorithm because the output is a result of the difference of two positions. A noteworthy point in the velocity algorithm is the fact that $m_r$ disappears, which means that the velocity form operates without having to know the initial position of the device being controlled. The real advantage in using the velocity form of the algorithm is most prominent in process control where the digital controller is working in conjunction with an operator's manual control. If the operator manually controlled the process without the computer, the return to the positional form of control would necessitate the operator providing the information on the position ($m_r$) of the process. Failing to do so would cause a disturbance. Using the velocity form of control eliminates this disturbance and gives the controller what is typically called 'bumpless transfer'. Considering this feature, along with the fact that Smith considers this form 30% more accurate than the approximate integral-derivative method, the algorithm developed for use in the microprocessor will be in the velocity form.

Algorithm Programming

Rewriting Equation 4.3 for $m_n$ and collecting terms of $e_i$ yields

$$m_n = K_c(1 + \frac{T}{T_i} + \frac{T_d}{T})e_n + K_c(-1 - \frac{2T_d}{T})e_{n-1} + K_c(\frac{T_d}{T})e_{n-2} + m_{n-1} \qquad (4.4)$$

The algorithm form programmed on the microprocessor control system is

$$m_n = Ae_n + Be_{n-1} + Ce_{n-2} + Dm_{n-1} + Em_{n-2} \tag{4.5}$$

therefore, applying Equation 4.4 to the microprocessor requires that

$$A = K_c(1 + \frac{T}{T_i} + \frac{T_d}{T}) \tag{4.6}$$

$$B = K_c(-1- \frac{2T_d}{T}) \tag{4.7}$$

$$C = K_c(\frac{T_d}{T}) \tag{4.8}$$

$$D = 1.0 \tag{4.9}$$

$$E = 0.0 \tag{4.10}$$

In Equation 4.4, the second past value of the output, $m_{n-2}$, is not necessary to process the digital PID algorithm. It is included in the programming to allow the system more versatility in processing other types of second order digital filters and algorithms.

The entire controller system (including the control algorithm) was programmed on the microprocessor by use of the assembly language constructed by Control Logic, Inc. An explanation of the assembly language mnemonics and an assembly listing are found in the Appendix. This particular microprocessor had absolutely no software support as compared to other real-time high language level minicomputers and microprocessors. For this reason, the creation of special subroutines was necessary to handle normal routine events

such as teletype communications and double precision arithmetic.

The system is basically constructed as a set of subroutines supporting a main section of programming that is responsible for controlling and timing the execution of the subroutines. A general flowchart showing the logical progression of the system if shown in Figure 4.2. The teletype is used to input the five coefficients and the input reference value. At anytime during the execution of the real-time portion of the program, a command from the teletype will allow the input reference value or both the coefficients and the input reference value to be reinitialized. In the real-time portion of the program, the algorithm is processed and applied to the system at regular intervals determined by the interrupt structure. The first step in processing the algorithm is the appropriate summations of the multiplications of the past values of error and algorithm output by their respective coefficients. Upon receiving an interrupt signal, the feedback output signal from the plant system is sampled through the SHA-ADC combination. This sampled value is subtracted from the input reference value to yield an error signal. The error is multiplied by its respective coefficient, and the product is summed with the result of the aforementioned coefficient calculations. The final result is then applied to the plant system via the DAC. The past values are then updated and the process is repeated.

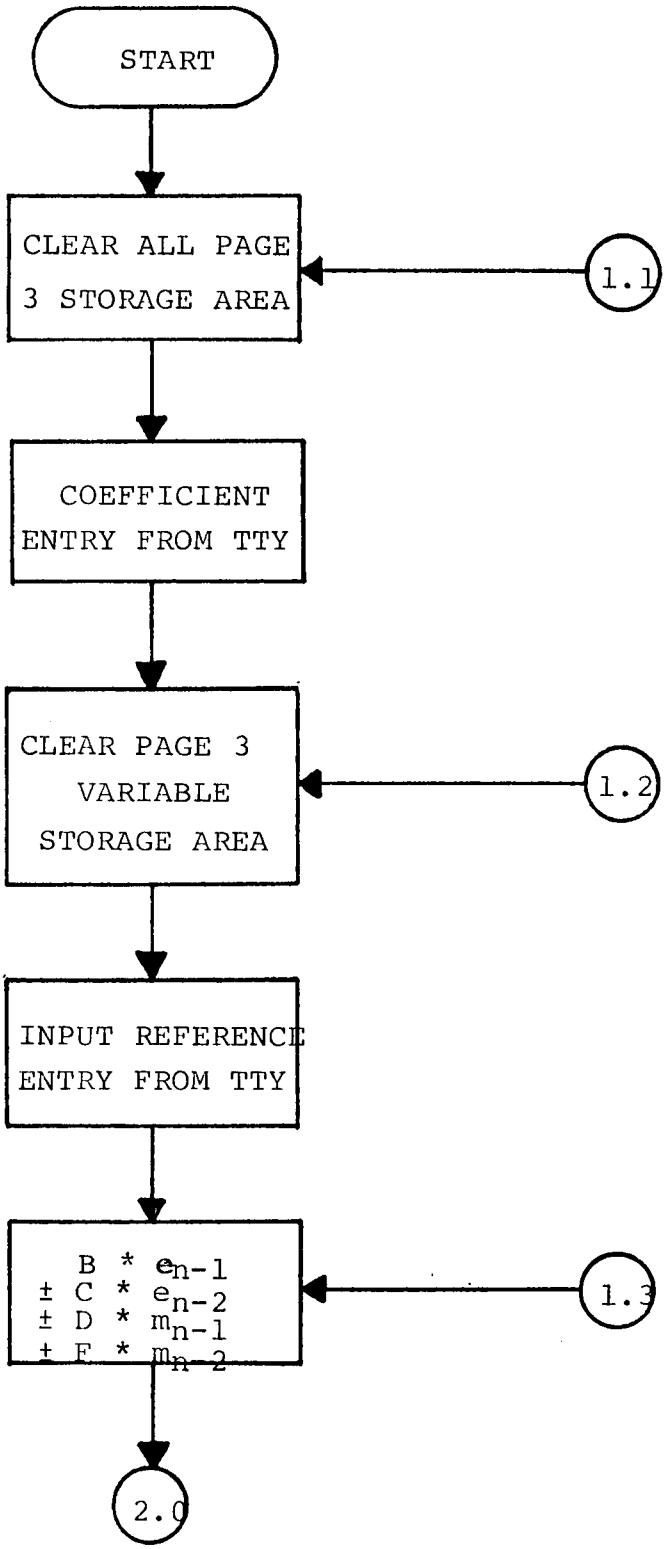



Figure 4.2. Flow chart of microprocessor controller.

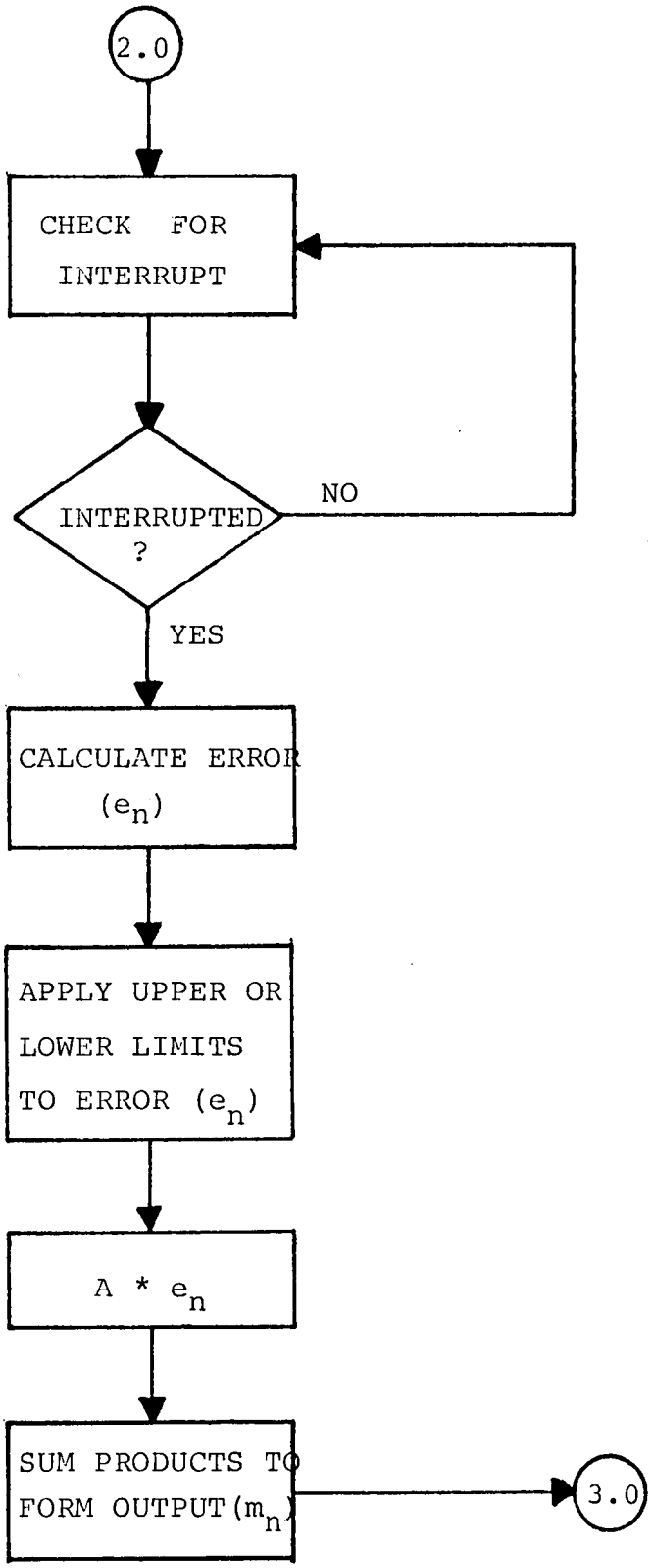



Figure 4.2. (continued)

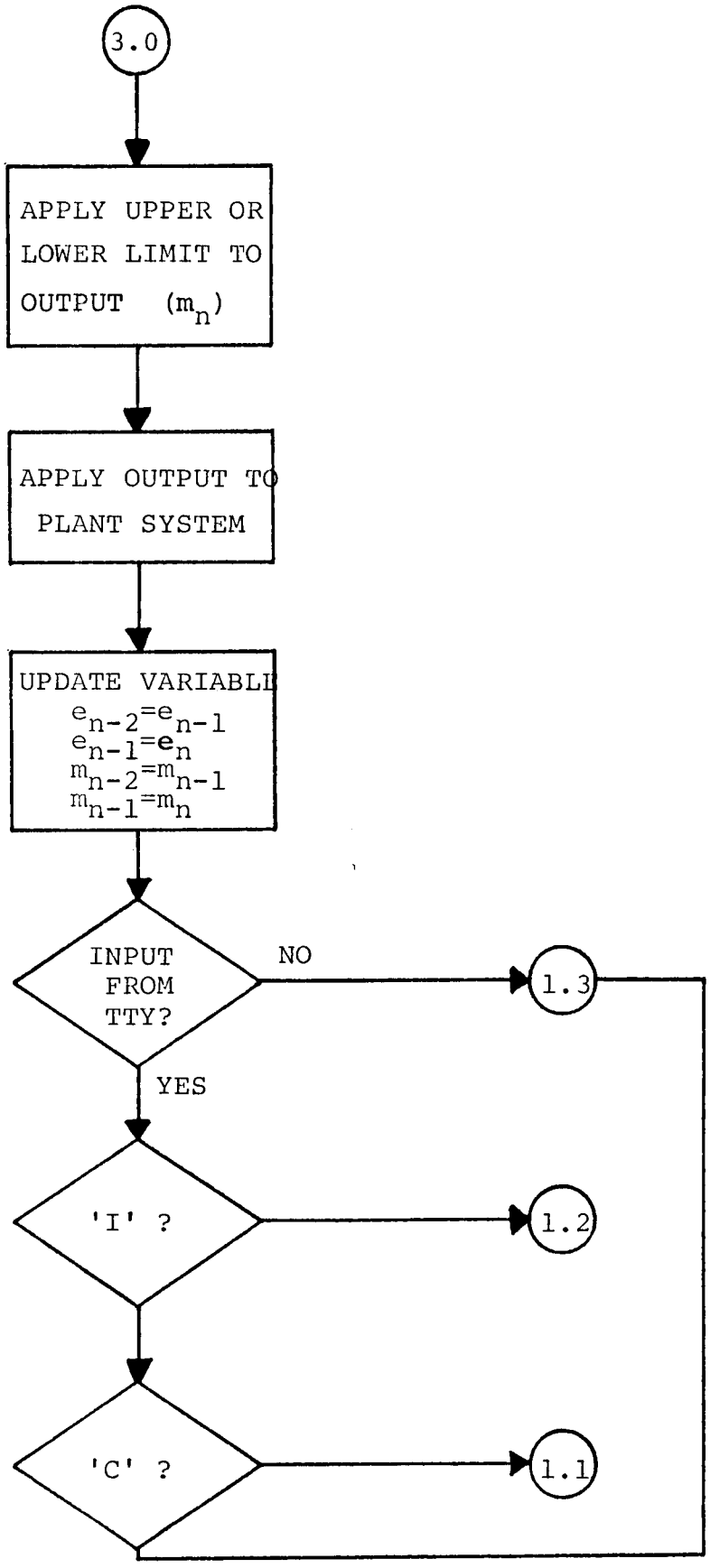



Figure 4.2. (continued)

It should be noted that the basic philosophy behind the real-time section of programming is to maintain a minimal amount of coding between the time that the feedback signal is sampled and the algorithm result calculated and applied to the system. A close examination of the assembly listing in the Appendix will show that there are no instructions after the interrupt that could not have been done before the interrupt. As previously mentioned in Chapter III, this optimal form of programming to minimize real-time delay is necessary to insure against instability of the overall system.

The two subroutines executed after the interrupt implement two subtle, but very critical factors. The first factor involves the way signals are represented by eight bits. The ADC and DAC limit these signals between -5 volts and +5 volts for a 10-volt range. To provide maximum resolution, the binary eight-bit representation of these signals matches the 10-volt range exactly. A problem therefore exists when the feedback signal is subtracted from the input reference value to form the error value. In calculating the error value, it is possible to obtain a result that is too large or too small to be represented by the eight-bit representation used by the program. For example, if the input reference value corresponds to +3 volts, and if the feedback signal is at -3 volts, then the resulting error value would correspond to +6 volts which is above the +5 volt maximum

limit. For this reason it is necessary to recognize when an out-of-limits error value occurs and limit that value accordingly. This is accomplished by subroutine ALIN, whose assembly listing and general flowchart can be found in Appendix A.

The second subtlety occurs after the entire algorithm has been calculated. When the sixteen-bit products in the algorithm are summed, the result is also a sixteen-bit number. Depending on the values of the coefficients and signals, it is possible to calculate a result that is above or below the limits of the DAC's ±5 volt range. Therefore, this value must be checked and limited if necessary. This is accomplished by subroutine ROLL, whose assembly and general flowchart can be found in Appendix A.

The implications of these two problems weigh heavily upon the overall performance of the control system. In both cases, the net result is a limitation on the size of the signal given to the plant system. When working with a slow responding plant system, it may not be possible to achieve fast rise times. It should not, however, affect the steady-state error.

## Results

The microprocessor control system was applied to a plant system as shown in Figure 4.3. The plant system was

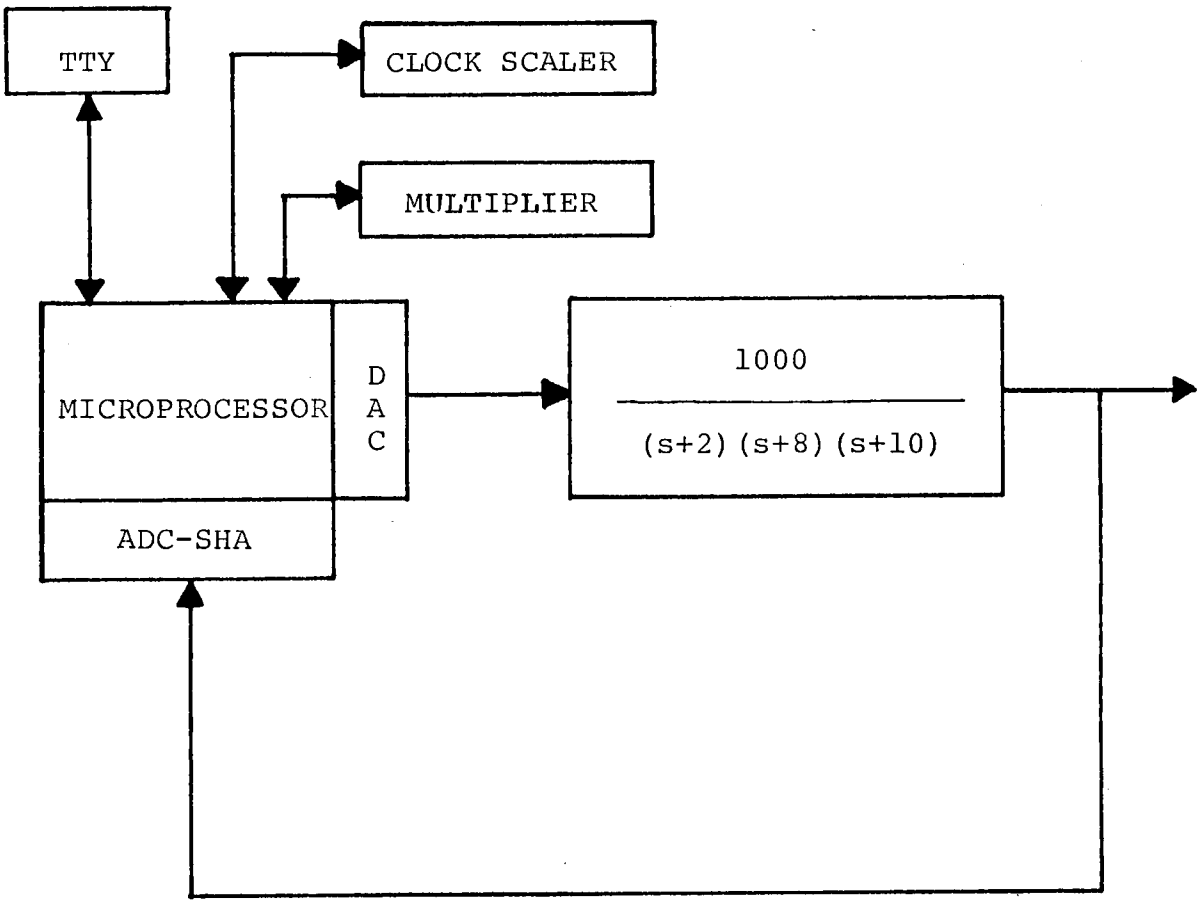



Figure 4.3. Microprocessor controller as applied to the plant system.

given sufficient gain to yield an unfavorable oscillatory condition bordering on marginal stability. A type zero system was chosen to give a finite steady-state error. Figure 4.4 shows its dynamic response to a step input of 2 volts.

There are several different methods for calculating the gain coefficients for the PID controller. The basic philosophy behind the initial coefficient values is to bring the system's dynamic response to such a level that a minimal amount of further adjustment is necessary. Depending on the plant system's characteristics, the use of these techniques reduces the time required for finding "optimum" controller parameters. The method incorporated for the microprocessor control system is based on two characteristics of the plant system's open-loop response to a step input (5). These two characteristics are the delay time after the input is applied and the time constant associated with the exponential rise of the open-loop response. In other words, one assumes that the open-loop transfer function is of the form

$$G_p(s) = \frac{Ke^{-\theta s}}{1+\tau s} \tag{4.11}$$

The values of K, $\theta$ and $\tau$ are either estimated from the measured step response or from the actual parameters associated with the plant transfer function

$$G_p(s) = \frac{1000}{(s+2)(s+8)(s+10)} \tag{4.12}$$

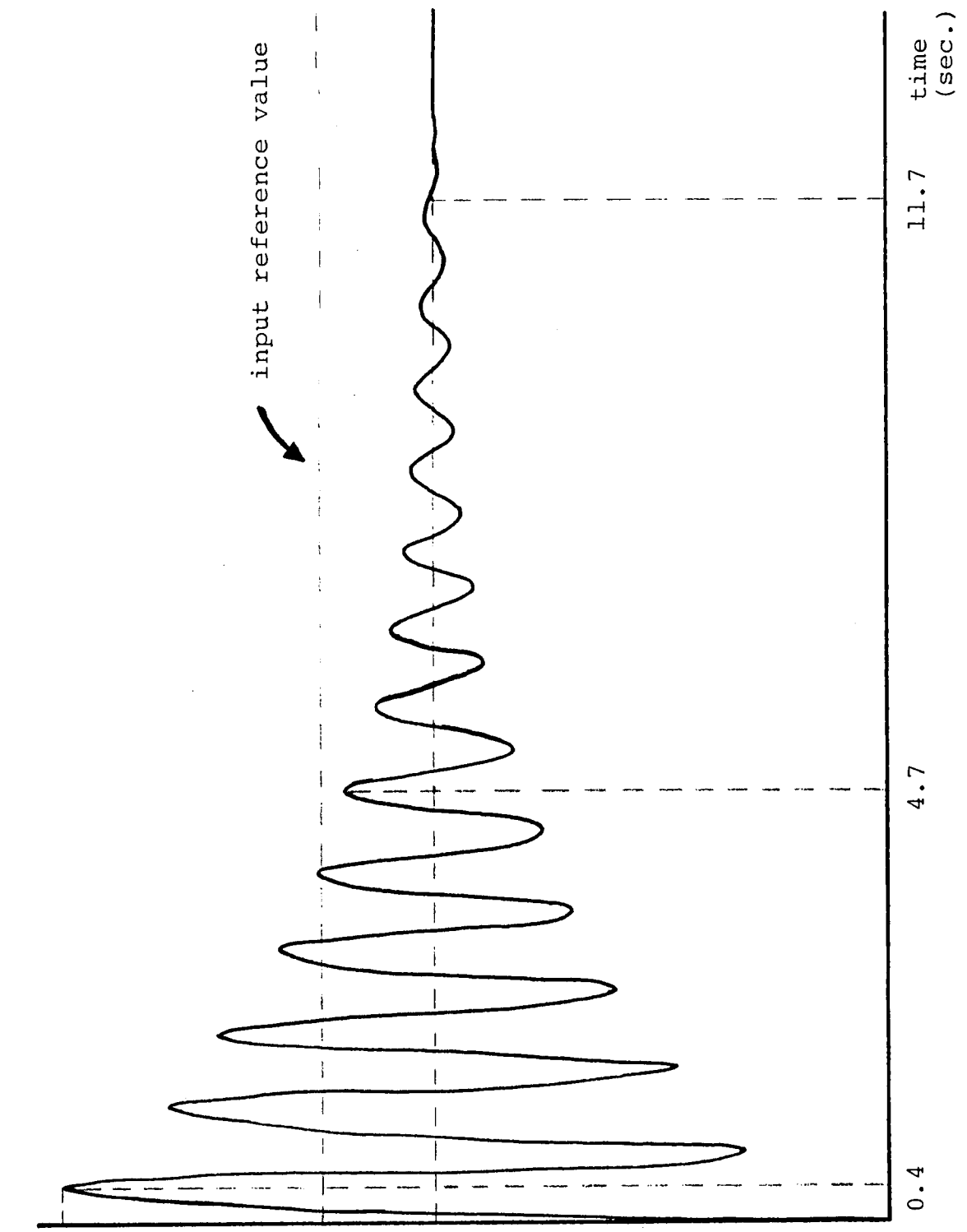



Figure 4.4. Dynamic response of uncontrolled system for a 2.0 voltage step change.

Following the procedure outlined by Smith (5), calculations for the controller parameters are made as follows.

For the proportional mode:

$$KK_c = A_1\left(\frac{\theta}{\tau}\right)^{B_1} \tag{4.13}$$

For the integral or reset mode:

$$\frac{\tau}{T_i} = A_2 + B_2\left(\frac{\theta}{\tau}\right) \tag{4.14}$$

For the derivative or rate mode:

$$\frac{T_d}{\tau} = A_3\left(\frac{\theta}{\tau}\right)^{B_3} \tag{4.15}$$

In the above expressions

$G_p$ = open-loop system transfer function

$K$ = gain of the plant system, $G_p(s)$

$K_c$ = controller proportional gain

$T_i$ = controller reset time

$T_d$ = controller rate time

$\tau$ = estimated time constant of $G_p(s)$

$\theta$ = estimated ideal delay of $G_p(s)$

$A_i$, $B_i$ = table parameters given in Smith(5)

$(i = 1,2,3)$

The parameters $A_i$ and $B_i$ are selected from a table of values corresponding to the type of controller, system excitation, and performance index. For this application, the cofficients were selected on the basis of set-point change and minimization of the integral absolute error (IAE).

Using a delay time of 0.125 seconds and a time constant of 0.6 seconds, the controller parameters are

$$K_c = 0.679 \tag{4.16}$$

$$T_i = 0.842 \tag{4.17}$$

$$T_d = 0.0498 \tag{4.18}$$

These values are then used in Equations 4.6, 4.7, and 4.8 to calculate the algorithm coefficients, which are

$$A = 1.096$$

$$B = 1.353$$

$$C = 0.338$$

After using these coefficients as a starting point, the following algorithm is judged to provide the best system dynamic response:

$$m_n = (0.790)(e_n) - (0.938)(e_{n-1}) + (0.234)(e_{n-2}) + (1.0)(m_{n-1}) \tag{4.19}$$

The response generated by the use of these coefficients is shown in Figure 4.5. In comparing the uncontrolled system to the controlled system, the most obvious accomplishment is the improvement on the stability of the system. The controlled system takes 3 or 4 oscillations to reach steady-state as compared to approximately 15 for the uncontrolled system. Referring to Table 4.1, it can be seen that the overshoot is slightly improved, the steady-state error is greatly improved, and the rise time remained about the same.

The microprocessor control system has been digitally simulated via the Continuous System Modeling Program (CSMP) on an IBM 370 computer. The CSMP model contains all of the known resolution losses throughout the controller. Figure 4.6 shows the areas in the calculation process where the

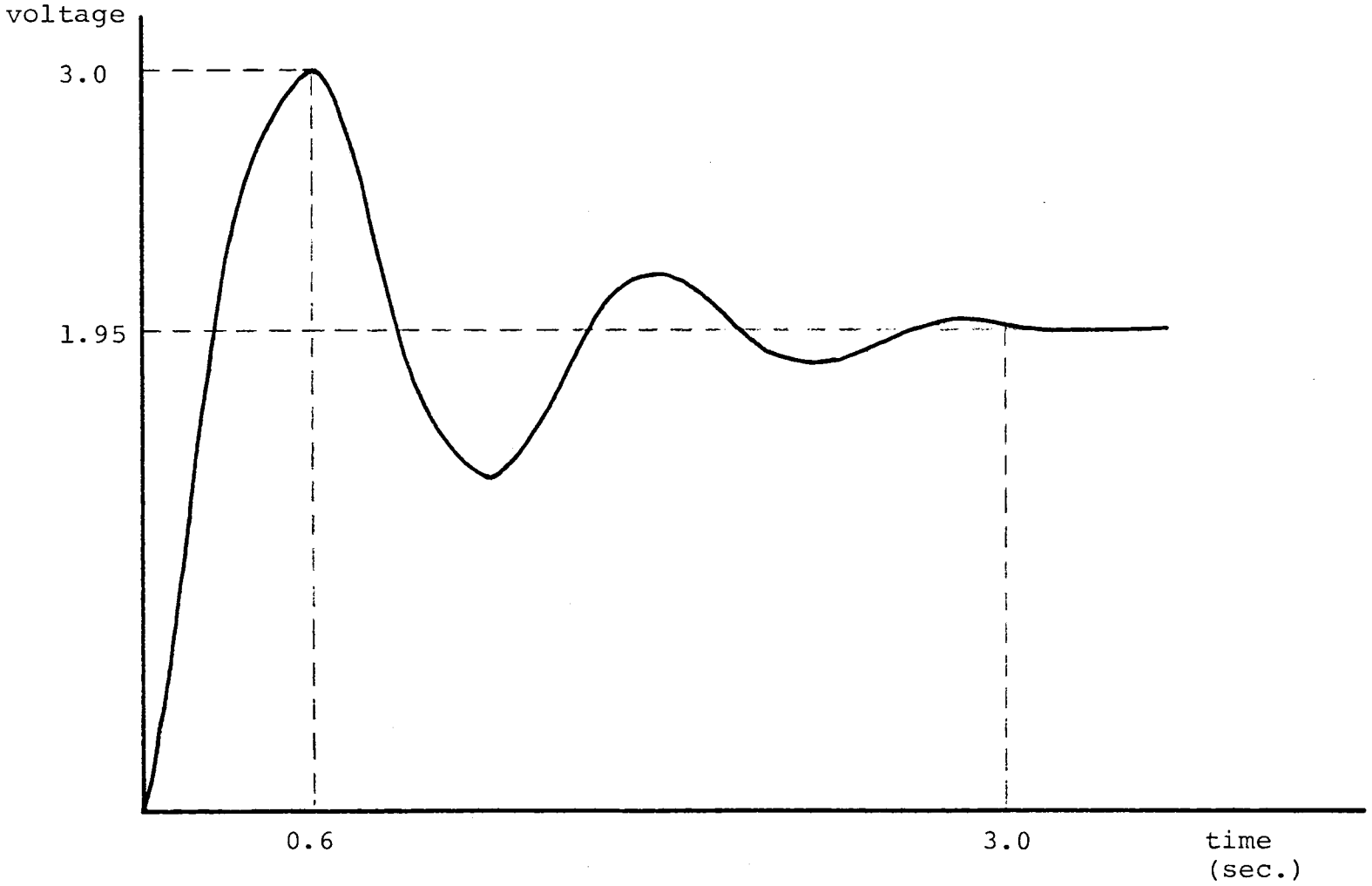



Figure 4.5. Dynamic response of controlled system for a 2.0 voltage step change.

TABLE 4.1

COMPARISON OF SIMULATION TO ACTUAL RESULTS

| | Microprocessor System | | CSMP Simulation | |
|---|---|---|---|---|
| | Uncontrolled | Controlled | Uncontrolled | Controlled |
| % Overshoot | 86.7% | 53.8% | 80% | 35% |
| Steady-state | 1.7 Volts | 1.95 Volts | 1.75 Volts | 2.0 Volts |
| % Steady-state error | 15% | 2.5% | 12.5% | 0% |
| Time to peak | 0.4 Sec. | 0.6 Sec. | 0.43 Sec. | 0.6 Sec. |
| Settling time | 11.7 Sec. | 3.0 Sec. | 12.0 Sec. | 3.0 Sec. |

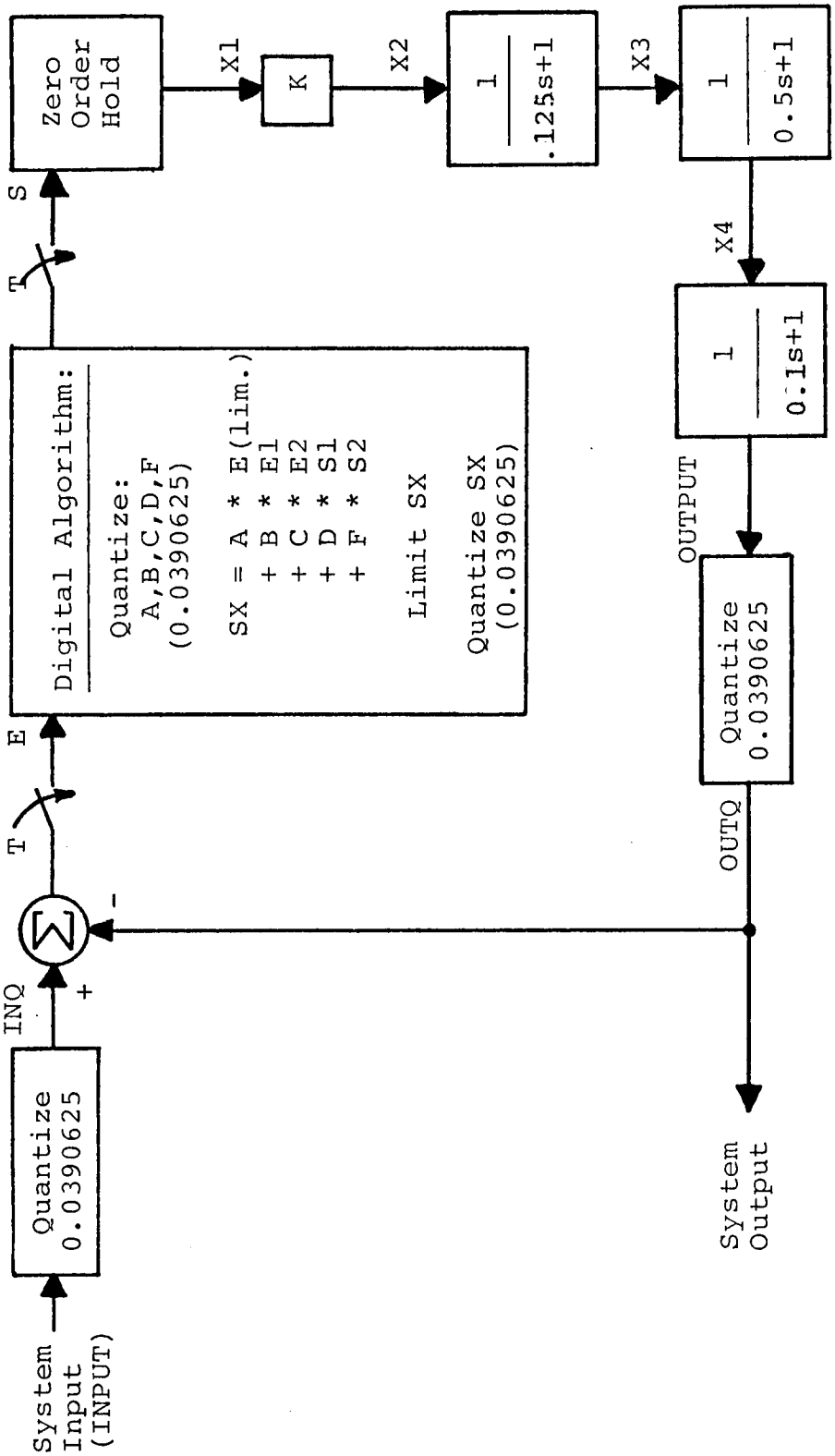



Figure 4.6. Block diagram of CSMP simulation.

losses occur. These losses can be attributed to the quantities being limited to eight bits in the microprocessor. Applying eight bits to a 10-volt range yields a resolution of 0.0390625 volts, which is programmed as a quantization function in the CSMP model. The CSMP programs for the uncontrolled and controlled systems are shown in Figure 4.7 and Figure 4.8, respectively. The results of the simulation of the uncontrolled and controlled systems are shown in Figure 4.9 and Figure 4.10, respectively. Table 4.1 compares the results of the CSMP simulation of the actual results obtained with the microprocessor. In both cases, the waveform is similar to that experienced with the microprocessor. Also, in both cases, the rise time and settling time of the waveforms agree very closely. The one major difference is in the overshoot. The microprocessor experiences slightly more overshoot than the simulation produces.

```
      ****CONTINUOUS SYSTEM MODELING PROGRAM****

      ***PROBLEM INPUT STATEMENTS***

INITIAL
      E1=0.0
      E2=0.0
      S1=0.0
      S2=0.0
      PARAM K=6.25
DYNAMIC
      INP=STEP(0.0)
      INPUT=2.0*INP
      INQ=QNTZR(0.0390625,INPUT)
      E=INQ-OUTQ
      OUTQ=QNTZR(0.0390625,OUTPUT)
      OUTPUT=REALPL(0.0,0.1,X4)
      X4=REALPL(0.0,0.5,X3)
      X3=REALPL(0.0,0.125,X2)
      X2=X1*K
      X1=ZHOLD(A1,S)
      A1=IMPULS(0.0,0.1)
PROCEDURE  S=DUMMY(E,E1,E2,S1,S2,A1)
      IF(A1.NE.1.0)GO TO 10
      IF(KEEP.NE.1)GO TO 10
      IF(E.GT.5.0)E=5.0
      IF(E.LT.-5.0)E=-5.0
      SXX=1.0*E
      SX=QNTZR(0.0390625,SXX)
      IF(SX.GT.5.0)SX=5.0
      IF(SX.LT.-5.0)SX=-5.0
      S=SX
      E2=E1
      E1=E
      S2=S1
      S1=S
10    CONTINUE
ENDPRO
METHOD RKSFX
TIMER FINTIM=5.0, OUTDEL=0.1, DELT=0.001
PRTPLT OUTPUT
```

Figure 4.7. CSMP program for uncontrolled system.

```
      ****CONTINUOUS SYSTEM MODELING PROGRAM****

      ***PROBLEM INPUT STATEMENTS***

INITIAL
      E1=0.0
      E2=0.0
      S1=0.0
      S2=0.0
      PARAM K=6.25
DYNAMIC
      INP=STEP(0.0)
      INPUT=2.0*INP
      INQ=QNTZR(0.0390625,INPUT)
      E=INQ-OUTQ
      OUTQ=QNTZR(0.0390625,OUTPUT)
      OUTPUT=REALPL(0.0,0.1,X4)
      X4=REALPL(0.0,0.5,X3)
      X3=REALPL(0.0,0.125,X2)
      X2=X1*K
      X1=ZHOLD(A1,S)
      A1=IMPULS(0.0,0.1)
PROCEDURE  S=DUMMY(E,E1,E2,S1,S2,A1)
      IF(A1.NE.1.0)GO TO 10
      IF(KEEP.NE.1)GO TO 10
      IF(E.GT.5.0)E=5.0
      IF(E.LT.-5.0)E=-5.0
      Z1=0.79*E
      Z2=QNTZR(0.0390625,Z1)
      Z3=0.938*E1
      Z4=QNTZR(0.0390625,Z3)
      Z5=0.234*E2
      Z6=QNTZR(0.0390625,Z5)
      Z7=1.0*S1
      Z8=QNTZR(0.0390625,Z7)
      SXX=Z2-Z4+Z6+Z8
      SX=QNTZR(0.0390625,SXX)
      IF(SX.GT.5.0)SX=5.0
      IF(SX.LT.-5.0)SX=-5.0
      S=SX
      E2=E1
      E1=E
      S2=S1
      S1=S
10    CONTINUE
ENDPRO
METHOD RKSFX
TIMER FINTIM=5.0, OUTDEL=0.1, DELT=0.001
PRINTPLOT OUTPUT
```

Figure 4.8. CSMP program for controlled system.

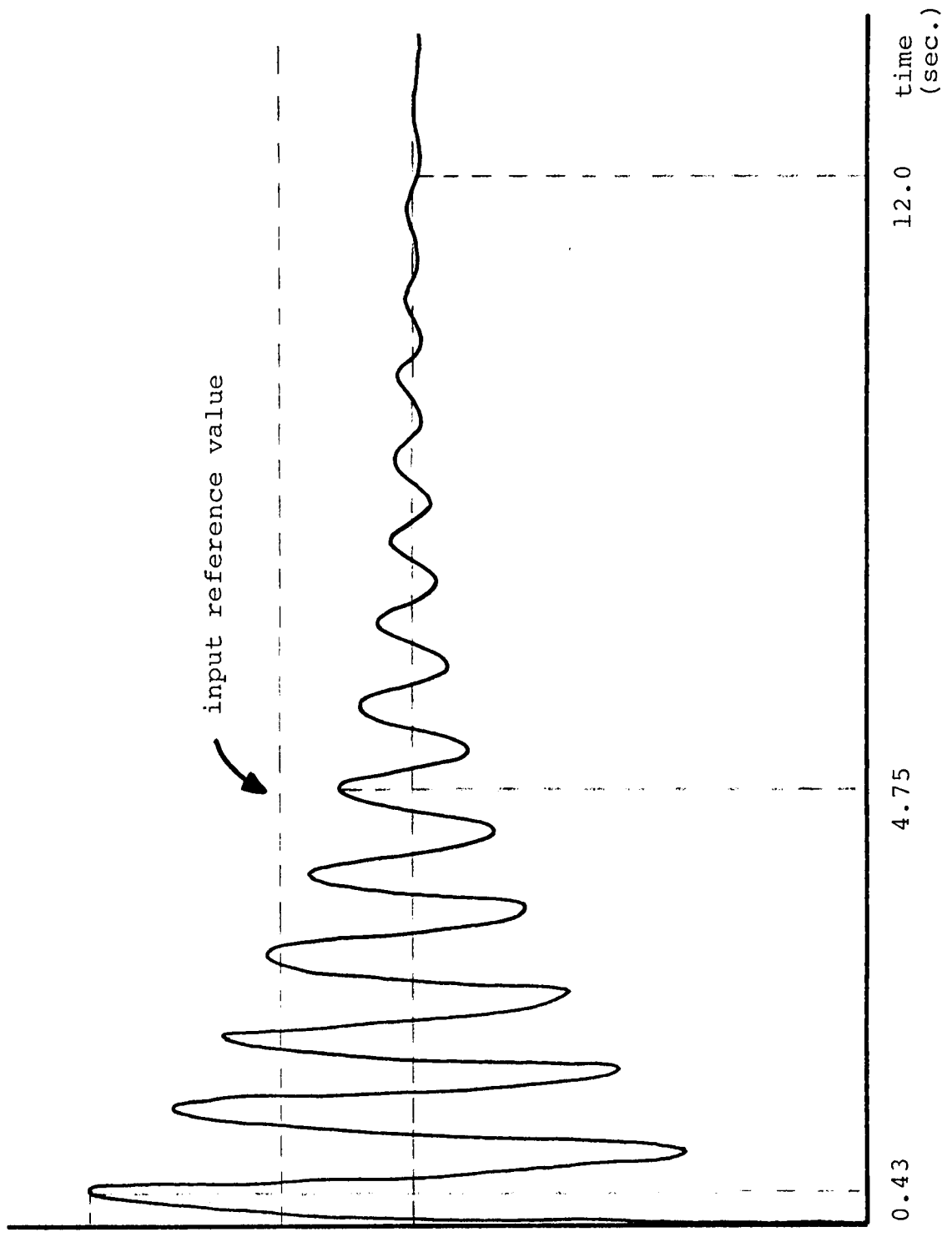



Figure 4.9. CSMP dynamic response of uncontrolled system for a 2.0 voltage step change.

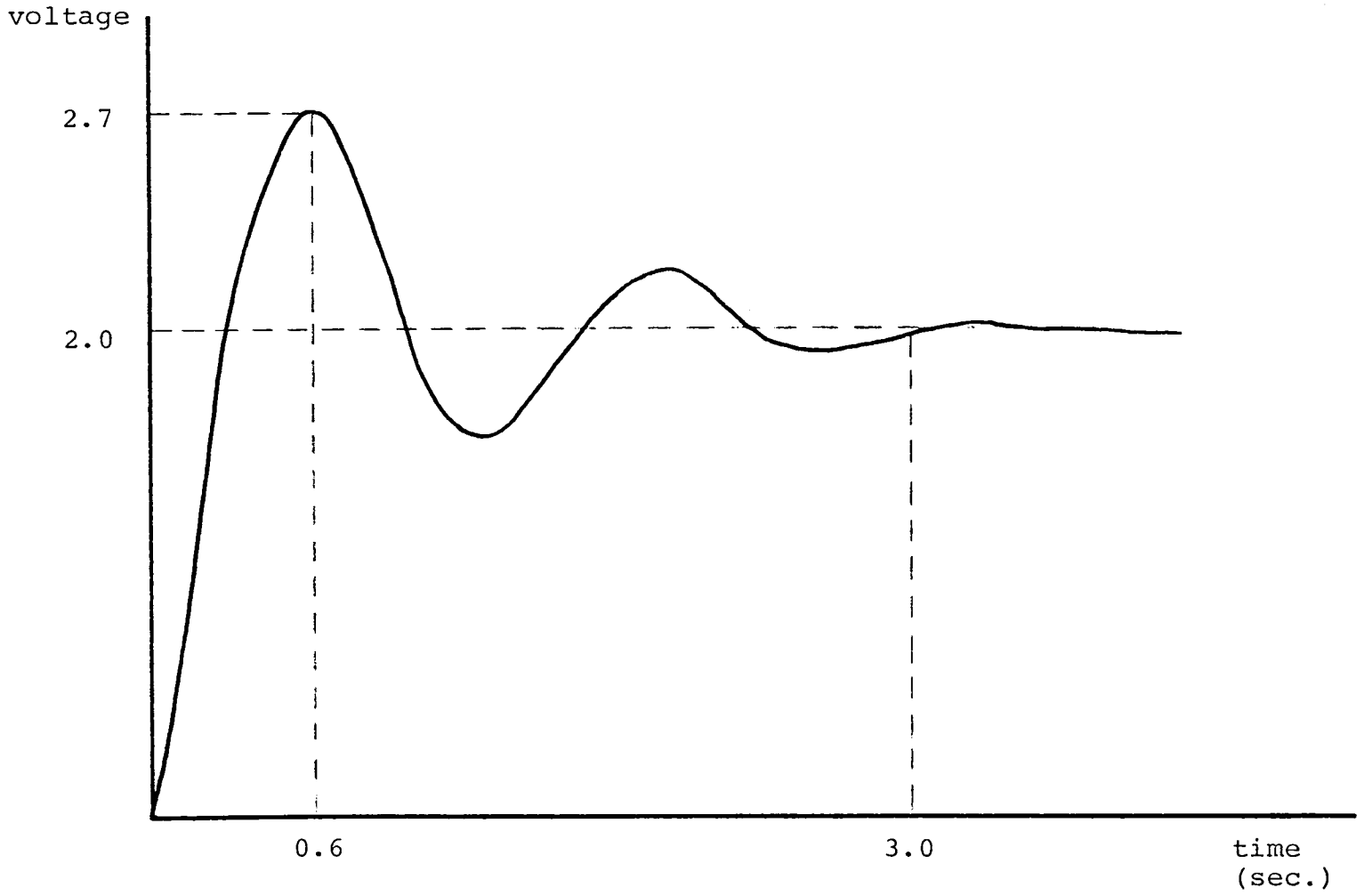



Figure 4.10. CSMP dynamic response of controlled system for a 2.0 voltage step change.

# CHAPTER V

## CONCLUSIONS AND RECOMMENDATIONS

A procedure for applying a digital control algorithm to a typical plant system via a microprocessor has been presented. The basic operating characteristics of the microprocessor, as well as the design and fabrication of specialized digital hardware, has also been presented. In summary, a versatile microprocessor control system capable of processing a second order digital algorithm for the use of process control or signal filtering has been realized.

Three performance limitations of the microprocessor control system have been discussed:

1. The nature of the software speed makes it impossible to operate with the 0.001 second sampling rate. It is also necessary to derive the control algorithm with a finite real-time delay in order to operate at a sampling rate of 0.01 second. Sampling rates of 0.1 and 1.0 second are the only rates selectable from the clock scaling circuit that can operate without restructuring the software or the control algorithm.
2. The use of eight-bit words to store the difference of the feedback signal and the input reference voltage makes it necessary to limit the error signal to a value between -5 volts and +5 volts. Reorganization of the software to allow the error value to vary over

the entire range between -10 volts and +10 volts would be at the expense of the addition of considerable amount of time in the main control loop.

3. The output voltage range of the DAC is limited to a value between -5 volts and +5 volts. The effect of this limitation on the performance of the microprocessor control system is the inability to apply large error signals to the plant system. In essence, the system's rise time that should theoretically occur with some algorithms may actually occur slower in the real application.

An example of a feedback control application in which a microprocessor may be used has been realized with satisfactory results. The system has been simulated using the IBM/370 Continuous Systems Modeling Program producing results similar to those experienced with the microprocessor.

Further study should be oriented towards the improvement of the microprocessor in the areas of calculation power, speed, and resolution. A microprocessor with improved performance in these areas could be applied more readily to the processing of a digital control algorithm.

The microprocessor, like any computer, provides a control system that is reliable, flexible, and accurate. While in many cases the microprocessor is more expensive than analog circuitry, its digital compatibility with supervisory computers increases the efficiency of the overall

control system. In addition, it can execute tasks that analog circuitry is not capable of performing. With its continued decrease in cost, the microprocessor will enjoy a continual growth in applications to the field of direct digital control.

# LIST OF REFERENCES

# LIST OF REFERENCES


1. T. R. Blakeslee, Digital Design with Standard MSI and LSI, New York, N.Y.: John Wiley & Sons, 1975.

2. J. A. Cadzow, Discrete-time Systems, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1973.

3. V. W. Eveleigh, Introduction to Control Systems Design, New York, N.Y.: McGraw-Hill Book Company, 1972.

4. J. B. Peatman, The Design of Digital Systems, New York, N.Y.: McGraw-Hill Book Company, 1972.

5. C. L. Smith, Digital Computer Process Control, Scranton, Pa.: International Textbook Company, 1972.

6. C. Weitzman, Minicomputer Systems, Englewood Cliffs, N.J.: Prentice-Hall, Incorporated, 1974.

7. L. E. Cannon and P. S. Kreager, "Using a Microprocessor: A Real-Life Application, Part 2 - Software," Computer Design, 14 (October, 1975), 81.

8. D. C. Collins, "Microprocessor Design and Applications," Computer, 8 (October, 1975), 20.

9. R. K. Jurgen, "The Microprocessor: In the Driver's Seat?," IEEE Spectrum, 12 (June, 1975), 73.

10. T. A. Laliotis, "Microprocessors Present and Future," Computer, 7 (July, 1974), 20.

11. J. Little and A. T. Thomas, "Operator's Console Considerations in Microprocessor System Design," 14 (November, 1975), 87.

12. D. R. Nelson, "Microprocessor Applications," Computer, 7 (August, 1974), 19.

13. J. Rattner, J. C. Cornet, and M. E. Hoff, Jr., "Bipolar LSI Computer Elements Usher In New Era of Digital Design," Electronics, (September, 1974), 89.

14. L-Series Microprocessor System Manuals, Control Logic, Incorporated, Natick, Mass., 1973.

15. Continuous System Modeling Program User's Manual (360A-CS-16X), IBM, 1973.

# APPENDIX

# APPENDIX

## PROGRAM DOCUMENTATION

This appendix gives an overall description of the basic organization of the microprocessor control program, including explanations of the operation of all subroutines. A complete assembly listing of the program and subroutines is included.

### Main Program Organization

The first section of the program clears to zero the area in memory where the intermediate data (such as coefficients, input reference, add or subtract flags, and present and past signal values) are stored.

The next section of the program indicates to the user that the coefficients may be entered by printing a "C" on the teletype. The user then may enter the first coefficient in the format of an integer between -128 and +127, a decimal point, and a fractional part with up to three significant digits. A negative sign must be typed first for negative coefficients. The absence of a sign will be assumed to mean a positive coefficient. Typing a carriage return completes the coefficient. A "C" is printed on the teletype for each subsequent coefficient entry. The coefficients must be entered in the order A, B, C, D, and F, and all five must be entered.

At this point, the program clears those areas in memory where the present and past values of the signals are stored. This is done so that the program can be re-entered and the input reference changed without having to repeat the input of the coefficients. The program indicates its readiness to accept the input reference by printing an "I" on the teletype. The user then enters the input reference by typing a two-digit number followed by a carriage return. A decimal point is assumed by the program between the two digits. For negative input references, a negative sign must precede the two digits, but positive coefficients need no sign indication. Entries must be limited to between -5.0 and +5.0.

Both the coefficient entry and the input coefficient entry are converted to numbers easily stored in memory by means of table look-up procedures. These procedures are explained in the documentation of subroutines INPUT, INT, and FRC.

Upon receiving the carriage return after the input reference entry, the real-time portion of the program is entered. The first operation performed is the calculation of B, C, D, and E coefficient-signal products and their sum. The method by which this is accomplished is detailed in the documentation of subroutine MUL. The program then waits for an interrupt from the clock scaling circuitry. When the interrupt occurs, the interrupt service routine enables

the microprocessor to continue execution of the real-time portion of the program. The error signal is then calculated, and the resulting magnitude is limited to a value between -5.0 and +5.0 by subroutine ALIN. The result is multiplied by the A coefficient and added or subtracted, according to the status of the respective add/subtract flag, with the summation of the other coefficient-signal products. This final result is then truncated to eight bits. After limiting the truncated result to a value between -5.0 and +5.0, the algorithm result is applied to the plant system via the DAC.

The next section of the program updates the past values of the feedback signal and the algorithm output. The second past value is replaced by the first past value, and the first past value is replaced by the present value.

At this point, the program checks to see if any communication has been entered from the teletype. If there is none, the program repeats the real-time portion of the program. If communication has been attempted, the program brings the data in. If a "C" has been typed, the program will be executed from the beginning, allowing re-initialization of the coefficients and the input reference. If an "I" has been typed, the program will be entered just before the entry of the input reference, allowing the input reference to be changed without changing the coefficients. If the communication is neither a "C" nor an "I," the real-time portion of the program is repeated.

## Subroutine Documentation

Subroutine IRET. Subroutine IRET sets a flag (register C) to signal the main program that an interrupt has occurred. The subroutine is initiated by the main program when it polls the interrupt structure for indications that any deferred interrupts have occurred.

Subroutine ROLL. Subroutine ROLL is executed by the main program at the completion of the processing of the algorithm. Its purpose is to limit the sixteen-bit result of the algorithm to an eight-bit number corresponding to the ±5 volt DAC limits. A general flowchart of the logic of the subroutine is shown in Figure A.1.

Subroutine INPUT. Subroutine INPUT takes two numbers from the teletype and creates an octal number to represent the corresponding input reference voltage. For example, an input of 10 from the teletype is converted to an octal number of 032. The octal conversion is accomplished by the summation of an octal value from two look-up tables corresponding to the two digits from the teletype. These tables are shown in Tables A.1 and A.2. A minus sign typed previous to the two numbers will in effect tell the subroutine to perform a two's complement on the result of the conversion.

Subroutine READ. This subroutine waits for a character to come from the teletype, receives the character

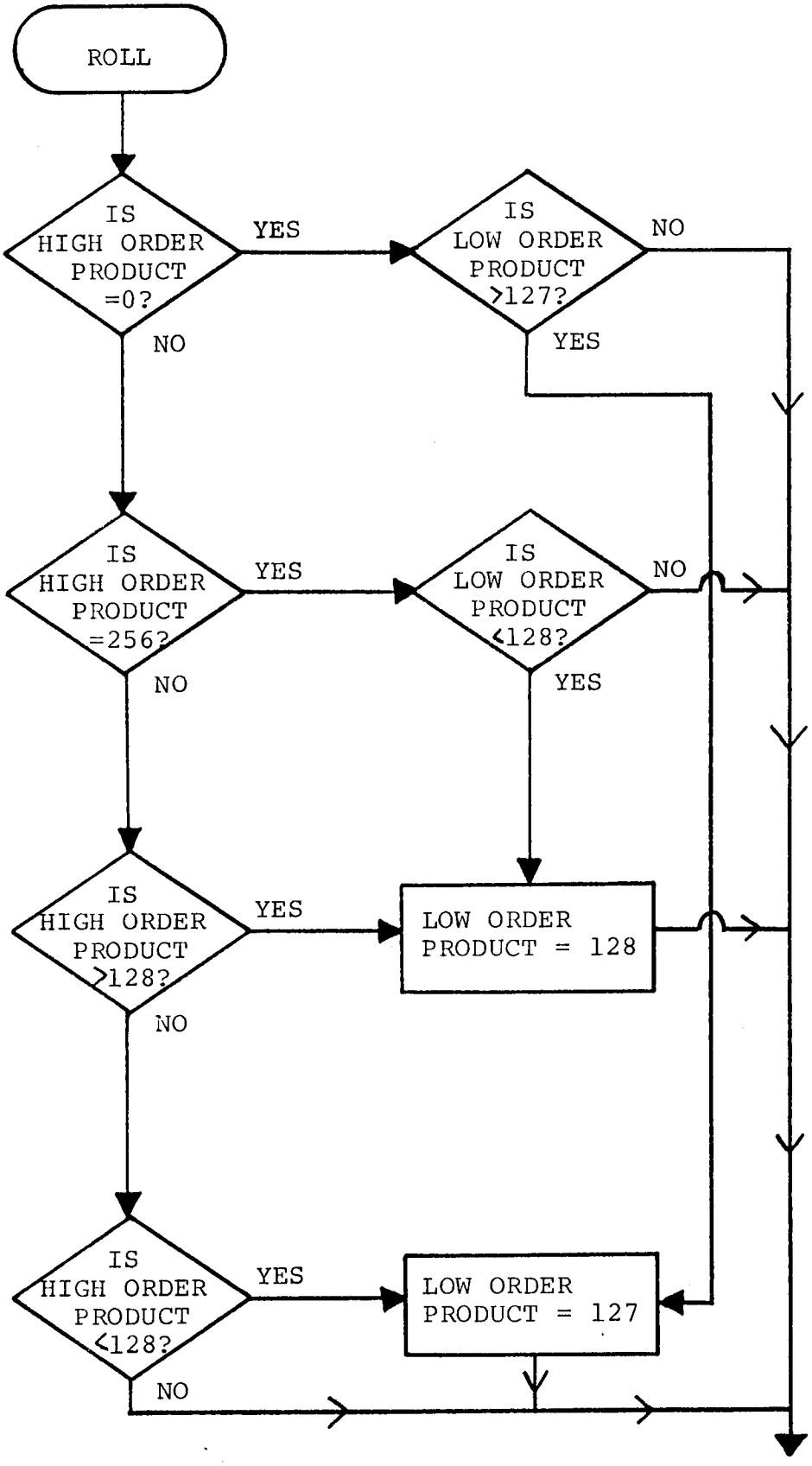



Figure A.1. Flow chart of subroutine ROLL.

TABLE A.1.

DECIMAL INTEGER TO OCTAL

| Decimal Digit | Octal Conversion |
|---|---|
| 0 | 000 |
| 1 | 001 |
| 2 | 002 |
| 3 | 003 |
| 4 | 004 |
| 5 | 005 |
| 6 | 006 |
| 7 | 007 |
| 8 | 010 |
| 9 | 011 |

TABLE A.2.

DECIMAL TENTHS TO OCTAL

| Decimal Digit | Octal Conversion |
|---|---|
| 0 | 000 |
| 1 | 032 |
| 2 | 063 |
| 3 | 115 |
| 4 | 146 |
| 5 | 177 |
| 6 | 232 |
| 7 | 263 |
| 8 | 315 |
| 9 | 346 |

in register A, and strips its transmission bit. The resulting seven-bit character is left in register B. The character is also sent back to the teletype to print the character that was sent. The subroutine is called by the main program as well as subroutines INPUT, INT and FRC.

Subroutine SEND. Subroutine SEND sends a character in register A to the teletype and does not terminate until that function is complete. It is called by the main program as well as subroutines READ and CRLF.

Subroutine CRLF. This subroutine sends a carriage return and a line feed to the teletype. It is executed from the main program.

Subroutine INT. This subroutine accepts the integer portion of the coefficient from the teletype and stores the octal equivalent in the memory storage area. The conversion to octal is done by the same type of table look-up procedure used in subroutine INPUT. The table used by INT may be found in Table A.1. Upon receiving a "period" character from the teletype, the subroutine terminates. It is executed from the main program.

Subroutine FRC. Subroutine FRC accepts the fractional portion of the coefficient from the teletype

and stores the octal equivalent in the memory storage area. The conversion to octal is done by the same type of table look-up procedure used in subroutine INPUT. The tables used by FRC are the same as Tables A.2, A.3, and A.4. Upon receiving a "carriage return" character from the teletype the subroutine terminates. It is executed from the main program.

Subroutine MUL. Subroutine MUL does the multiplication of the first and second past values by their respective coefficients. It also does the double-precision summation of those products. It is called by the main program.

Subroutine ALIN. Subroutine ALIN checks the error calculated from the feedback signal and the input reference value. If it is out of limits of eight-bit capability, it sets the value to the appropriate maximum or minimum limit. A general flowchart of the logic behind the subroutine is shown in Figure A.2. It is called by the main program.

## Program Assembly Listing

The following is an assembly listing of the main control program and all of the supporting subroutines. The address is broken down into an octal number representing the page and an octal number representing the location

TABLE A.3.

DECIMAL HUNDREDTHS TO OCTAL

| Decimal Digit | Octal Conversion |
|---|---|
| 0 | 000 |
| 1 | 003 |
| 2 | 005 |
| 3 | 010 |
| 4 | 012 |
| 5 | 015 |
| 6 | 017 |
| 7 | 022 |
| 8 | 024 |
| 9 | 027 |

TABLE A.4.

DECIMAL THOUSANDTHS TO OCTAL

| Decimal Digit | Octal Conversion |
|---|---|
| 0 | 000 |
| 1 | 000 |
| 2 | 001 |
| 3 | 001 |
| 4 | 001 |
| 5 | 001 |
| 6 | 002 |
| 7 | 002 |
| 8 | 002 |
| 9 | 002 |

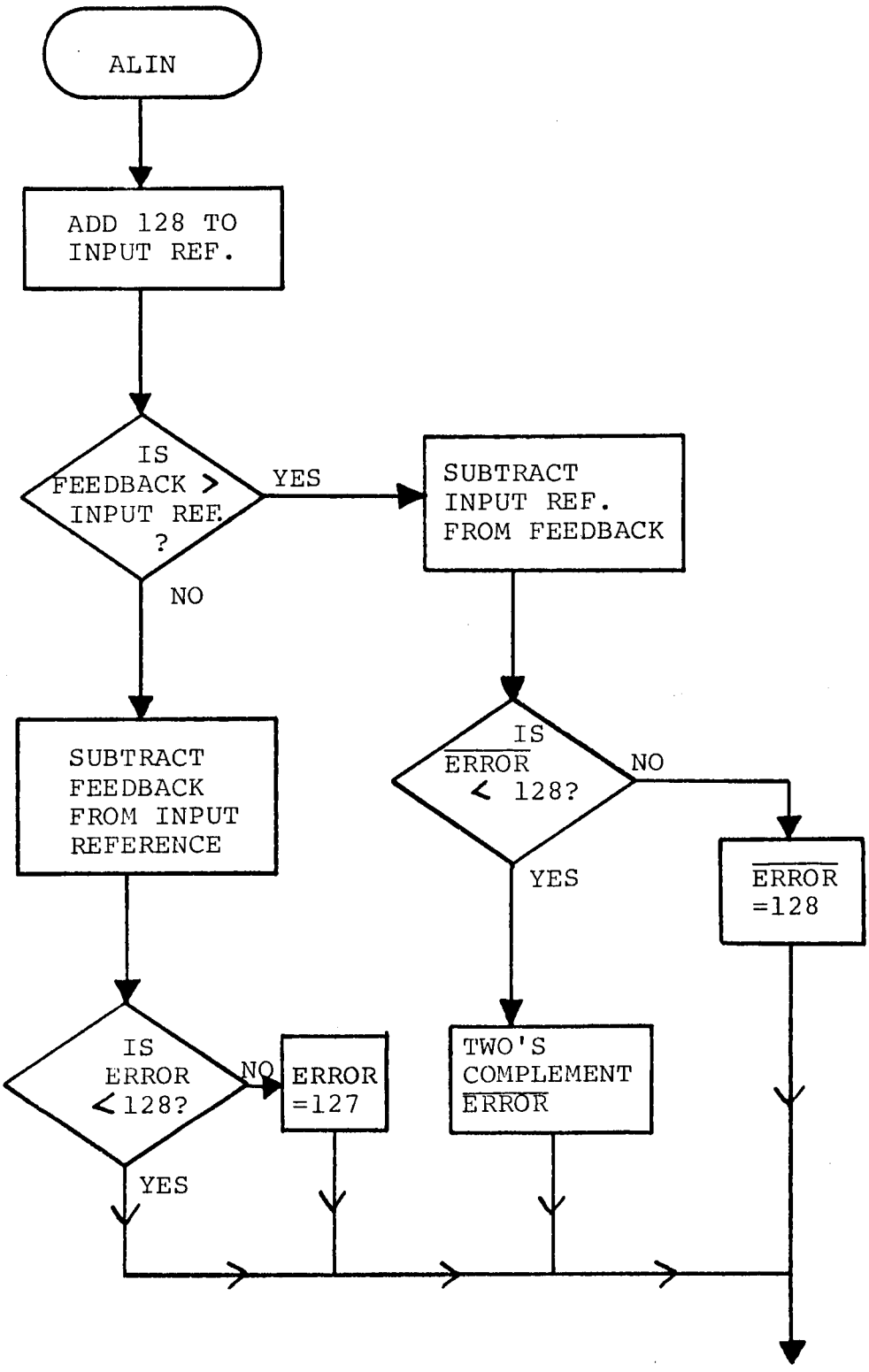



Figure A.2. Flow chart of Subroutine ALIN.

within that page. All of the Op Codes and numbers appearing in the Mnemonics are octal. A description of each instruction mnemonic is given in Table A.5. Table A.6 lists the locations of variables in the memory storage area.

TABLE A.5.

MICROPROCESSOR INSTRUCTION SET

| Mnemonic[a] | Code[b] | Description |
|---|---|---|
| Lds | 11 DDD SSS | Load register d with the contents of register s. |
| LdM | 11 DDD 111 | Load register d with the contents of memory. |
| LMs | 11 111 SSS | Load memory with the contents of register s. |
| LdI | 00 DDD 110<br>BB BBB BBB | Load register d with data B----B. |
| LMI | 00 111 110<br>BB BBB BBB | Load memory with data B----B. |
| INd | 00 DDD 000 | Increment the contents of register d (d≠ A). |
| DCd | 00 DDD 001 | Decrement the contents of register d (d≠ A). |
| JMP | 01 000 100<br>LL LLL LLL<br>00 HHH HHH | Unconditionally jump to memory address H----HL----L. |
| JFc | 01 0CC 000<br>LL LLL LLL<br>00 HHH HHH | Jump to memory address H----HL----L if flip-flop c is false. Otherwise, execute the next instruction in sequence. |
| JTc | 01 1CC 000<br>LL LLL LLL<br>00 HHH HHH | Jump to memory address H----HL----L if flip-flop c is true. Otherwise, execute the next instruction in sequence. |
| CAL | 01 000 110<br>LL LLL LLL<br>00 HHH HHH | Unconditionally call the subroutine at memory address H----HL----L. Save the current address (up one level in stack). |

TABLE A.5. (continued)

| Mnemonic[a] | Code[b] | Description |
|---|---|---|
| CFc | 01 OCC 010<br>LL LLL LLL<br>00 HHH HHH | Call the subroutine at memory address H----HL----L if the flip-flop c is false, and save the current address (up one level in stack). Otherwise, execute the next instruction in sequence. |
| CTc | 01 1CC 010<br>LL LLL LLL<br>00 HHH HHH | Call the subroutine at memory address H----HL----L if the flip-flop c is true, and save the current address (up one level in stack). Otherwise, execute the next instruction in sequence. |
| RET | 00 000 111 | Unconditionally return (down one level in stack). |
| RFc | 00 0CC 011 | Return (down one level in stack) if the flip-flop c is false. Otherwise, execute the next instruction in sequence. |
| RTc | 00 1CC 011 | Return (down one level in stack) if the flip-flop c is true. Otherwise, execute the next instruction in sequence. |
| RST | 00 AAA 101 | Call subroutine at memory address 00AAA (up one level in stack). |
| INP | 01 00M MM1 | Read the contents of the selected input port MMM into the accumulator. |
| OUT | 01 RRM MM1 | Write the contents of the accumulator into the selected output port RRMMM, RR≠100. |

TABLE A.5. (continued)

| Mnemonic[a] | Code[b] | Description |
|---|---|---|
| ADs<br>ADM<br>ADI | 10 000 SSS<br>10 000 111<br>BB BBB BBB | Add the contents of register s, memory or data B----B to the accumulator. An overflow sets the carry flip-flop. |
| ACs<br>ACM<br>ACI | 10 001 SSS<br>10 001 111<br>00 001 100<br>BB BBB BBB | Add the contents of register s, memory or data B----B to the accumulator with carry. An overflow sets the carry flip-flop. |
| SUs<br>SUM<br>SUI | 10 010 SSS<br>10 010 111<br>00 010 100<br>BB BBB BBB | Subtract the contents of register s, memory or data B----B from the accumulator. An underflow sets the carry flip-flop. |
| SBs<br>SBM<br>SBI | 10 011 SSS<br>10 011 111<br>00 011 100<br>BB BBB BBB | Subtract the contents of register s, memory or data B----B from the accumulator with carry. An underflow sets the carry flip-flop. |
| NDs<br>NDM | 10 100 SSS<br>10 100 111<br>00 100 100<br>BB BBB BBB | Compute the logical AND of the contents of register s, memory or data B----B with the accumulator. |
| XRs<br>XRM<br>XRI | 10 101 SSS<br>10 101 111<br>00 101 100<br>BB BBB BBB | Compute the EXCLUSIVE OR of the contents of register s, memory or data B----B with the accumulator. |
| ORs<br>ORM<br>ORI | 10 110 SSS<br>10 110 111<br>00 110 100<br>BB BBB BBB | Compute the INCLUSIVE OR of the contents of register s, memory or data B----B with the accumulator. |
| Cps<br>CPM<br>CPI | 10 111 SSS<br>10 111 111<br>00 111 100<br>BB BBB BBB | Compare the contents of register s, memory or data B----B with the accumulator. Accumulator is unchanged. If equal, set zero flip-flop. If |

TABLE A.5. (continued)

| Mnemonic[a] | Code[b] | Description |
|---|---|---|
| | | the accumulator is less than s, set carry flip-flop; if greater, clear the carry. |
| RLC | 00 000 010 | Rotate the contents of the accumulator left one bit. |
| RRC | 00 001 010 | Rotate the contents of the accumulator right one bit. |
| RAR | 00 011 010 | Rotate the contents of the accumulator right one bit through the carry. |
| RAL | 00 010 010 | Rotate the contents of the accumulator left one bit through the carry. |
| HLT | 11 111 111 | Enter the STOPPED state and remain there until interrupted. |

[a]Memory is addressed by registers H (high-order address) and L (low-order address).
s = source register
d = destination register
c = condition flip-flop

[b]SSS = source register
DDD = destination register
where A = 000, B = 001, C = 010, D = 011, E = 100, H = 101, L = 110
CC = condition flip-flop
where c(carry) = 00, Z(zero result) = 01, s(sign) = 10, p(parity) = 11
HHH HHH = high-order address
LL LLL LLL = low-order address

TABLE A.6.

MEMORY STORAGE AREA (PAGE THREE)

| Location | Constant |
|---|---|
| 001 | Fraction - coefficient F |
| 002 | Add/subtract flag (F*S2) |
| 003 | S2 |
| 004 | Integer - coefficient F |
| 005 | Fraction - coefficient D |
| 006 | Add/subtract flag (D*S1) |
| 007 | S1 |
| 010 | Integer - coefficient D |
| 011 | Fraction - coefficient C |
| 012 | Add/subtract flag (C*E2) |
| 013 | E2 |
| 014 | Integer - coefficient C |
| 015 | Fraction - coefficient B |
| 016 | Add/subtract flag (B*E1) |
| 017 | E1 |
| 020 | Integer - coefficient B |
| 021 | Fraction - coefficient A |
| 022 | Add/subtract flag (A*E) |
| 023 | Input reference value |
| 024 | Integer - coefficient A |
| 025 | E |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 001 000 | 056 | OVER | LHI 003 | Set memory pointer to |
| 001 001 | 003 | | | variable storage area, |
| 001 002 | 066 | | LLI 024 | last location. |
| 001 003 | 024 | | | |
| 001 004 | 250 | | XRA | Clear Reg. A. |
| 001 005 | 370 | MEMCL | LMA | Load memory with zero. |
| 001 006 | 061 | | DCL | Decrement memory pointer. |
| 001 007 | 110 | | JFZ MEMCL | Is memory pointer 0? |
| 001 010 | 005 | | | If no, go to MEMCL. |
| 001 011 | 001 | | | If yes, continue. |
| 001 012 | 026 | | LCI 022 | Initialize alternate |
| 001 013 | 022 | | | pointer. |
| 001 014 | 046 | COEF | LEI 000 | Initialize table pointer |
| 001 015 | 000 | | | to Table 1. |
| 001 016 | 006 | | LAI 303 | Load Reg. A with |
| 001 017 | 303 | | | ASCII "C." |
| 001 020 | 106 | | CAL SEND | Print "C" on TTY. |
| 001 021 | 065 | | | |
| 001 022 | 002 | | | |
| 001 023 | 106 | | CAL READ | Read char. from TTY. |
| 001 024 | 050 | | | |
| 001 025 | 002 | | | |
| 001 026 | 301 | | LAB | Load char. in Reg. A. |
| 001 027 | 362 | | LLC | Use alternate pointer. |
| 001 030 | 074 | | CPI 055 | Is character a "-?" |
| 001 031 | 055 | | | |
| 001 032 | 110 | | JFZ C2 | |
| 001 033 | 046 | | | If no, go to C2. |
| 001 034 | 001 | | | If yes, continue. |
| 001 035 | 076 | | LMI 001 | Set add/subt. flag. |
| 001 036 | 001 | | | |
| 001 037 | 106 | | CAL READ | Read char. from TTY. |
| 001 040 | 050 | | | |
| 001 041 | 002 | | | |
| 001 042 | 301 | | LAB | Load char. in Reg. A. |
| 001 043 | 104 | | JMP C3 | Go to C3. |
| 001 044 | 050 | | | |
| 001 045 | 001 | | | |
| 001 046 | 076 | C2 | LMI 000 | Reset Add/Subt. flag. |
| 001 047 | 000 | | | |
| 001 050 | 020 | C3 | INC | Set alternate pointer |
| 001 051 | 020 | | INC | to integer address. |
| 001 052 | 106 | | CAL INT | Obtain integer |
| 001 053 | 110 | | | portion of coefficient. |
| 001 054 | 002 | | | |
| 001 055 | 021 | | DCC | Set alternate pointer |
| 001 056 | 021 | | DCC | to fraction address. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 001 057 | 021 | | DCC | |
| 001 060 | 046 | | LEI 012 | Set table pointer to |
| 001 061 | 012 | | | Table 2. |
| 001 062 | 106 | | CAL FRC | Obtain fractional |
| 001 063 | 151 | | | portion of coefficient. |
| 001 064 | 002 | | | |
| 001 065 | 106 | | CAL CRLF | Send carriage return |
| 001 066 | 075 | | | and line feed to TTY. |
| 001 067 | 002 | | | |
| 001 070 | 021 | | DCC | Decrement alternate pointer. |
| 001 071 | 150 | | JTZ VCLR | Is alternate pointer 0? |
| 001 072 | 101 | | | If yes, go to VCLR. |
| 001 073 | 001 | | | If no, continue |
| 001 074 | 021 | | DCC | Set alternate pointer |
| 001 075 | 021 | | DCC | to next coefficient. |
| 001 076 | 104 | | JMP COEF | Go to COEF (repeat). |
| 001 077 | 014 | | | |
| 001 100 | 001 | | | |
| 001 101 | 250 | VCLR | XRA | Clear Reg. A. |
| 001 102 | 066 | | LLI 023 | Set memory pointer |
| 001 103 | 023 | | | to INPUT. |
| 001 104 | 370 | | LMA | Set INPUT to 0. |
| 001 105 | 066 | | LLI 017 | Set memory pointer |
| 001 106 | 017 | | | to E1. |
| 001 107 | 370 | | LMA | Set E1 to 0. |
| 001 110 | 066 | | LLI 013 | Set memory pointer |
| 001 111 | 013 | | | to E2. |
| 001 112 | 370 | | LMA | Set E2 to 0. |
| 001 113 | 066 | | LLI 007 | Set memory pointer |
| 001 114 | 007 | | | to S1. |
| 001 115 | 370 | | LMA | Set S1 to 0. |
| 001 116 | 066 | | LLI 003 | Set memory pointer |
| 001 117 | 003 | | | to S2. |
| 001 120 | 370 | | LMA | Set S2 to 0. |
| 001 121 | 006 | | LAI 311 | Load Reg. A with |
| 001 122 | 311 | | | ASCII "I." |
| 001 123 | 106 | | CAL SEND | Print "I" on TTY. |
| 001 124 | 065 | | | |
| 001 125 | 002 | | | |
| 001 126 | 106 | | CAL INPUT | Obtain Input Reference. |
| 001 127 | 200 | | | |
| 001 130 | 000 | | | |
| 001 131 | 250 | BCDF | XRA | Clear Reg. A. |
| 001 132 | 330 | | LDA | Clear Reg. D. |
| 001 133 | 340 | | LEA | Clear Reg. E. |
| 001 134 | 066 | | LLI 020 | Initialize memory |
| 001 135 | 020 | | | pointer. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 001 136 | 106 | | CAL MUL | Obtain B*E1±C*E2 |
| 001 137 | 203 | | | ±D*S1±F*S2. |
| 001 140 | 002 | | | |
| 001 141 | 066 | | LLI 024 | Initialize memory pointer. |
| 001 142 | 024 | | | |
| 001 143 | 307 | | LAM | Load signed-variable side |
| 001 144 | 127 | | OUT SEL11 | of multiplier w/coef.-Int. |
| 001 145 | 061 | | DCL | Decrement memory pointer. |
| 001 146 | 317 | | LBM | Load Reg. B w/Input Ref. |
| 001 147 | 061 | | DCL | Decrement memory pointer. |
| 001 150 | 026 | IRGO | LCI 000 | Clear interrupt flag. |
| 001 151 | 000 | | | |
| 001 152 | 117 | | INP SEL7 | Enable deferred interrupts. |
| 001 153 | 021 | | DCC | Decrement interrupt flag. |
| 001 154 | 110 | | JFZ IRGO | Is interrupt flag 0? |
| 001 155 | 150 | | | If no, go to IRGO. |
| 001 156 | 001 | | | If yes, continue. |
| 001 157 | 105 | | INP SEL2 | Start ADC conversion. |
| 001 160 | 111 | | INP SEL4 | Input feedback from ADC. |
| 001 161 | 106 | | CAL ALIN | Calculate error(E) |
| 001 162 | 322 | | | and apply upper or |
| 001 163 | 002 | | | limit as necessary. |
| 001 164 | 066 | | LLI 025 | Set memory pointer |
| 001 165 | 025 | | | to E address. |
| 001 166 | 370 | | LMA | Load E in memory. |
| 001 167 | 066 | | LLI 022 | Set memory pointer |
| 001 170 | 022 | | | to Add/Subt. flag. |
| 001 171 | 131 | | OUT SEL12 | Load E into multiplier. |
| 001 172 | 113 | | INP SEL5 | Input high order of product |
| 001 173 | 320 | | LCA | into Reg. C. |
| 001 174 | 115 | | INP SEL6 | Input low order of product |
| 001 175 | 310 | | LBA | into Reg. B. |
| 001 176 | 307 | | LAM | Add/Subt. flag into Reg. A. |
| 001 177 | 074 | | CPI 000 | Is Add/Subt. flag 0? |
| 001 200 | 000 | | | |
| 001 201 | 110 | | JFZ 1R2 | |
| 001 202 | 241 | | | If yes, go to IR2. |
| 001 203 | 001 | | | If no, continue. |
| 001 204 | 303 | | LAD | Low order sum into A. |
| 001 205 | 201 | | ADB | Add low order product. |
| 001 206 | 330 | | LDA | Load sum into D. |
| 001 207 | 304 | | LAE | High order sum into A. |
| 001 210 | 212 | | ACC | Add high order product. |
| 001 211 | 340 | | LEA | Load sum into E. |
| 001 212 | 061 | | DCL | Decrement memory pointer. |
| 001 213 | 307 | | LAM | Load signed-variable side |
| 001 214 | 127 | | OUT SEL11 | of multiplier w/Coef. frac. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 01 215 | 113 | | INP $\overline{\text{SEL5}}$ | Input high order of product |
| 001 216 | 310 | | LBA | into Reg. B. |
| 001 217 | 026 | | LCI 000 | Clear pseudo-high order. |
| 001 220 | 000 | | | |
| 001 221 | 022 | | RAL | Rotate sign bit into carry. |
| 001 222 | 100 | | JFC M13 | Is product positive? |
| 001 223 | 227 | | | If yes, go to M13. |
| 001 224 | 001 | | | If no, continue. |
| 001 225 | 026 | | LCI 377 | Propagate sign bit |
| 001 226 | 377 | | | through pseudo-high order. |
| 001 227 | 032 | M13 | RAR | Rotate carry back. |
| 001 230 | 303 | | LAD | Low order sum into A. |
| 001 231 | 201 | | ADB | Add high order product. |
| 001 232 | 330 | | LDA | Sum into D. |
| 001 233 | 304 | | LAE | High order sum into A. |
| 001 234 | 212 | | ACC | Add pseudo-high order. |
| 001 235 | 340 | | LEA | Sum into E. |
| 001 236 | 104 | | JMP 1R3 | Go to IR3. |
| 001 237 | 273 | | | |
| 001 240 | 001 | | | |
| 001 241 | 303 | IR2 | LAD | Low order sum into A. |
| 001 242 | 221 | | SUB | Subtract low order product. |
| 001 243 | 330 | | LDA | Difference into D. |
| 001 244 | 304 | | LAE | High order sum into A. |
| 001 245 | 232 | | SBC | Subtract high order product. |
| 001 246 | 340 | | LEA | Difference into E. |
| 001 247 | 061 | | DCL | Decrement memory pointer. |
| 001 250 | 307 | | LAM | Load signed-variable side |
| 001 251 | 127 | | OUT $\overline{\text{SEL11}}$ | of multiplier w/Coef. frac. |
| 001 252 | 113 | | INP $\overline{\text{SEL5}}$ | Input high order product |
| 001 253 | 310 | | LBA | into Reg. B. |
| 001 254 | 026 | | LCI 000 | Clear pseudo-high order. |
| 001 255 | 000 | | | |
| 001 256 | 022 | | RAL | Rotate sign bit into carry. |
| 001 257 | 100 | | JFC M14 | Is product positive? |
| 001 260 | 264 | | | If yes, go to M14. |
| 001 261 | 001 | | | If no, continue. |
| 001 262 | 026 | | LCI 377 | Propogate sign bit |
| 001 263 | 377 | | | through pseudo-high order. |
| 001 264 | 032 | M14 | RAR | Rotate carry back. |
| 001 265 | 303 | | LAD | Low order sum into A. |
| 001 266 | 221 | | SUB | Subtract high order product. |
| 001 267 | 330 | | LDA | Difference into D. |
| 001 270 | 304 | | LAE | High order sum into A. |
| 001 271 | 232 | | SBC | Subtract pseudo-high order. |
| 001 272 | 340 | | LEA | Difference into E. |
| 001 273 | 106 | IR3 | CAL ROLL | Apply upper or |
| 001 274 | 110 | | | lower limit to final |
| 001 275 | 000 | | | sum as necessary. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 001 276 | 303 | | LAD $\overline{\text{SEL10}}$ | Sum into D. |
| 001 277 | 125 | | OUT $\overline{\text{SEL10}}$ | Output to DAC. |
| 001 300 | 066 | | LLI 007 | Set memory pointer |
| 001 301 | 007 | | | to S1. |
| 001 302 | 317 | | LBM | S1 into Reg. B. |
| 001 303 | 370 | | LMA | S into S1. |
| 001 304 | 066 | | LLI 003 | Set memory pointer |
| 001 305 | 003 | | | to S2. |
| 001 306 | 371 | | LMB | S1 (old) into S2. |
| 001 307 | 066 | | LLI 017 | Set memory pointer |
| 001 310 | 017 | | | to E1. |
| 001 311 | 317 | | LBM | E1 into Reg. B. |
| 001 312 | 066 | | LLI 013 | Set memory pointer |
| 001 313 | 013 | | | to E2. |
| 001 314 | 371 | | LMB | E1 into E2. |
| 001 315 | 066 | | LLI 025 | Set memory pointer |
| 001 316 | 025 | | | to E. |
| 001 317 | 317 | | LBM | E into Reg. B. |
| 001 320 | 066 | | LLI 017 | Set memory pointer |
| 001 321 | 017 | | | to E1. |
| 001 322 | 371 | | LMB $\overline{\text{SEL1}}$ | E into E1. |
| 001 323 | 103 | | INP $\overline{\text{SEL1}}$ | Input I/O status lines. |
| 001 324 | 032 | | RAR | Rotate status 1 into carry. |
| 001 325 | 100 | | JFC BCDF | Does TTY have entry? |
| 001 326 | 131 | | | If no, go to BCDF. |
| 001 327 | 001 | | | If yes, continue. |
| 001 330 | 101 | | INP $\overline{\text{SEL0}}$ | Enable TTY and input. |
| 001 331 | 044 | | NDI 177 | Strip ASCII character. |
| 001 332 | 177 | | | |
| 001 333 | 074 | | CPI 111 | Is character "I?" |
| 001 334 | 111 | | | |
| 001 335 | 150 | | JTZ VCLR | |
| 001 336 | 101 | | | If yes, go to VCLR. |
| 001 337 | 001 | | | If no, continue. |
| 001 340 | 074 | | CPI 103 | Is character "C?" |
| 001 341 | 103 | | | |
| 001 342 | 150 | | JTZ OVER | |
| 001 343 | 000 | | | If yes, go to OVER. |
| 001 344 | 001 | | | If no, continue. |
| 001 345 | 104 | | JMP BCDF | Go to BCDF. |
| 001 346 | 131 | | | |
| 001 347 | 001 | | | |
| 000 040 | 020 | INTR | INC | Set interrupt flag. |
| 000 041 | 007 | | RET | Return. |
| 000 110 | 304 | ROLL | LAE | Load high order sum into A. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 000 111 | 074 | | CPI 000 | Is sum a legitimate |
| 000 112 | 000 | | | positive number? |
| 000 113 | 110 | | JFZ 1R4 | |
| 000 114 | 127 | | | If no, go to IR4. |
| 000 115 | 000 | | | If yes, continue. |
| 000 116 | 303 | | LAD | Load low order sum into A. |
| 000 117 | 074 | | CPI 177 | Has low order sum |
| 000 120 | 177 | | | exceeded upper limit? |
| 000 121 | 100 | | JFC 1R8 | |
| 000 122 | 164 | | | If yes, go to IR8. |
| 000 123 | 000 | | | If no, continue. |
| 000 124 | 104 | | JMP 1R9 | Go to IR9. |
| 000 125 | 166 | | | |
| 000 126 | 000 | | | |
| 000 127 | 074 | 1R4 | CPI 377 | Is sum a legitimate |
| 000 130 | 377 | | | negative number? |
| 000 131 | 110 | | JFZ IR5 | |
| 000 132 | 145 | | | If no, go to IR5. |
| 000 133 | 000 | | | If yes, continue. |
| 000 134 | 303 | | LAD | Load low order sum into A. |
| 000 135 | 074 | | CPI 200 | Has low order sum |
| 000 136 | 200 | | | exceeded lower limit? |
| 000 137 | 140 | | JTC 1R6 | |
| 000 140 | 152 | | | If yes, go to IR6. |
| 000 141 | 000 | | | If no, continue. |
| 000 142 | 104 | | JMP 1R9 | |
| 000 143 | 166 | | | |
| 000 144 | 000 | | | |
| 000 145 | 074 | 1R5 | CPI 200 | Is illegitimate sum |
| 000 146 | 200 | | | positive? |
| 000 147 | 140 | | JTC 1R7 | |
| 000 150 | 157 | | | If yes, go to IR7. |
| 000 151 | 000 | | | If no, continue. |
| 000 152 | 036 | 1R6 | LDI 200 | Load lower limit into D. |
| 000 153 | 200 | | | |
| 000 154 | 104 | | JMP 1R9 | Go to IR9. |
| 000 155 | 166 | | | |
| 000 156 | 000 | | | |
| 000 157 | 074 | 1R7 | CPI 200 | Is illegitimate sum |
| 000 160 | 200 | | | negative? |
| 000 161 | 100 | | JFC 1R9 | |
| 000 162 | 166 | | | If yes, go to IR9. |
| 000 163 | 000 | | | If no, continue. |
| 000 164 | 036 | 1R8 | LDI 177 | Load upper limit into D. |
| 000 165 | 177 | | | |
| 000 166 | 007 | 1R9 | RET | Return to call. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | | COMMENT |
|---|---|---|---|---|---|
| 000 200 | 066 | INPUT | LLI | 023 | Set memory pointer to |
| 000 201 | 023 | | | | INP. REF. location. |
| 000 202 | 076 | | LMI | 000 | Clear INP. REF. |
| 000 203 | 000 | | | | |
| 000 204 | 046 | | LEI | 000 | Clear sign flag. |
| 000 205 | 000 | | | | |
| 000 206 | 326 | | LCL | | Set alt. memory pointer. |
| 000 207 | 036 | | LDI | 012 | Set table pointer to |
| 000 210 | 012 | | | | table 2. |
| 000 211 | 106 | | CAL | READ | Read character from |
| 000 212 | 050 | | | | TTY. |
| 000 213 | 002 | | | | |
| 000 214 | 301 | | LAB | | Load character into A. |
| 000 215 | 074 | | CPI | 055 | Is character a "-?" |
| 000 216 | 055 | | | | |
| 000 217 | 110 | | JFZ | IN1 | |
| 000 220 | 227 | | | | If no, go to IN1. |
| 000 221 | 000 | | | | If yes, continue. |
| 000 222 | 040 | | INE | | Set sign flag. |
| 000 223 | 106 | INO | CAL | READ | Read character from TTY. |
| 000 224 | 050 | | | | |
| 000 225 | 002 | | | | |
| 000 226 | 301 | | LAB | | Load character into A. |
| 000 227 | 024 | IN1 | SUI | 060 | Convert ASCII to octal |
| 000 230 | 060 | | | | number. |
| 000 231 | 203 | | ADD | | Index table pointer. |
| 000 232 | 330 | | LDA | | Load pointer into D. |
| 000 233 | 360 | | LLA | | Set memory pointer to table. |
| 000 234 | 056 | | LHI | 002 | Set memory page to Z. |
| 000 235 | 002 | | | | |
| 000 236 | 307 | | LAM | | Load table value into A. |
| 000 237 | 056 | | LHI | 003 | Set memory page to 3. |
| 000 240 | 003 | | | | |
| 000 241 | 362 | | LLC | | Memory pointer to INP. REF. |
| 000 242 | 207 | | ADM | | Add table value to INP. REF. |
| 000 243 | 370 | | LMA | | Store sum in INP. REF. |
| 000 244 | 303 | | LAD | | Load table pointer into A. |
| 000 245 | 036 | | LDI | 024 | Set table pointer to |
| 000 246 | 024 | | | | table 3. |
| 000 247 | 074 | | CPI | 024 | Is previous table pointer |
| 000 250 | 024 | | | | in table 2? |
| 000 251 | 140 | | JTC | INO | |
| 000 252 | 223 | | | | If yes, go to INO. |
| 000 253 | 000 | | | | If no, continue. |
| 000 254 | 041 | | DCE | | Decrement sign flag. |
| 000 255 | 110 | | JFZ | IN2 | Is number to be negative? |
| 000 256 | 266 | | | | If no, go to IN2. |
| 000 257 | 000 | | | | If yes, continue. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 000 260 | 307 | | LAM | Load INP REF. into A. |
| 000 261 | 054 | | XRI 377 | Complement INP. REF. |
| 000 262 | 377 | | | |
| 000 263 | 004 | | ADI 001 | Add 1 to INP. REF. |
| 000 264 | 001 | | | |
| 000 265 | 370 | | LMA | Store result in INP. REF. |
| 000 266 | 106 | IN2 | CAL CRLF | Send carriage return |
| 000 267 | 075 | | | and line feed to TTY. |
| 000 270 | 002 | | | |
| 000 271 | 007 | | RET | Return to call. |
| | | | | |
| 002 000 | 000 | | | Table 1. |
| 002 001 | 001 | | | Decimal integer |
| 002 002 | 002 | | | to octal integer. |
| 002 003 | 003 | | | |
| 002 004 | 004 | | | |
| 002 005 | 005 | | | |
| 002 006 | 006 | | | |
| 002 007 | 007 | | | |
| 002 010 | 010 | | | |
| 002 011 | 011 | | | |
| | | | | |
| 002 012 | 000 | | | Table 2. |
| 002 013 | 032 | | | Decimal tenths |
| 002 014 | 063 | | | to octal fraction. |
| 002 015 | 115 | | | |
| 002 016 | 146 | | | |
| 002 017 | 177 | | | |
| 002 020 | 232 | | | |
| 002 021 | 263 | | | |
| 002 022 | 315 | | | |
| 002 023 | 346 | | | |
| 002 024 | 000 | | | Table 3. |
| 002 025 | 003 | | | Decimal hundredths |
| 002 026 | 005 | | | to octal fraction. |
| 002 027 | 010 | | | |
| 002 030 | 012 | | | |
| 002 031 | 015 | | | |
| 002 032 | 017 | | | |
| 002 033 | 022 | | | |
| 002 034 | 024 | | | |
| 002 035 | 027 | | | |
| | | | | |
| 002 036 | 000 | | | Table 4. |
| 002 037 | 000 | | | Decimal thousandths |
| 002 040 | 001 | | | to octal fraction. |
| 002 041 | 001 | | | |
| 002 042 | 001 | | | |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 002 043 | 001 | | | |
| 002 044 | 002 | | | |
| 002 045 | 002 | | | |
| 002 046 | 002 | | | |
| 002 047 | 002 | | | |
| 002 050 | 103 | READ | INP $\overline{\text{SEL1}}$ | Input I/O status lines. |
| 002 051 | 032 | | RAR | Rotate status 1 into carry. |
| 002 052 | 100 | | JFC READ | Does TTY have entry? |
| 002 053 | 050 | | | If no, go to READ. |
| 002 054 | 002 | | | If yes, continue. |
| 002 055 | 101 | | INP $\overline{\text{SEL0}}$ | Input data from TTY. |
| 002 057 | 044 | | NDI 177 | Strip transmission bit. |
| 002 059 | 177 | | | |
| 002 060 | 310 | | LBA | Load result into B. |
| 002 061 | 106 | | CAL SEND | Echo character on TTY. |
| 002 062 | 065 | | | |
| 002 063 | 002 | | | |
| 002 064 | 007 | | RET | Return to call. |
| 002 065 | 121 | SEND | OUT $\overline{\text{SEL8}}$ | Send data to TTY. |
| 002 066 | 103 | | INP $\overline{\text{SEL1}}$ | Input I/O status lines. |
| 002 067 | 032 | | RAR | Rotate status 1 into carry. |
| 002 070 | 032 | | RAR | Rotate status 2 into carry. |
| 002 071 | 043 | | RTC | Return if data has been |
| 002 072 | 104 | | JMP SEND | received, or go to |
| 002 073 | 066 | | | instruction after SEND |
| 002 074 | 002 | | | if it has not. |
| 002 075 | 006 | CRLF | LAI 215 | Load ASCII code for |
| 002 076 | 215 | | | carriage return into A. |
| 002 077 | 106 | | CAL SEND | Send to TTY. |
| 002 100 | 065 | | | |
| 002 101 | 002 | | | |
| 002 102 | 006 | | LAI 212 | Load ASCII code for |
| 002 103 | 212 | | | line feed into A. |
| 002 104 | 106 | | CAL SEND | Send to TTY. |
| 002 105 | 065 | | | |
| 002 106 | 002 | | | |
| 002 107 | 007 | | RET | Return to call |
| 002 110 | 024 | INT | SUI 060 | Convert ASCII to |
| 002 111 | 060 | | | octal number. |
| 002 112 | 204 | | ADE | Add index to table pointer. |
| 002 113 | 360 | | LLA | Set memory pointer. |
| 002 114 | 056 | | LHI 002 | Set memory page to 2. |
| 002 115 | 002 | | | |
| 002 116 | 307 | | LAM | Load table value into A. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 002 117 | 056 | | LHI 003 | Set memory page to 3. |
| 002 120 | 003 | | | |
| 002 121 | 362 | | LLC | Load memory pointer w/alt. |
| 002 122 | 207 | | ADM | Add table value to integer. |
| 002 123 | 370 | | LMA | Store result in integer loc. |
| 002 124 | 106 | | CAL READ | Read character from TTY. |
| 002 125 | 050 | | | |
| 002 126 | 002 | | | |
| 002 127 | 301 | | LAB | Load character into A. |
| 002 130 | 074 | | CPI 056 | Is character a "period?" |
| 002 131 | 056 | | | |
| 002 132 | 053 | | RTZ | If yes, return to call. |
| 002 133 | | | LBA | Load character into B. |
| 002 134 | 046 | | LEI 000 | Clear index. |
| 002 135 | 000 | | | |
| 002 136 | 307 | | LAM | Load signed-variable side |
| 002 137 | 127 | | OUT $\overline{\text{SEL11}}$ | of multiplier w/INPUT REF. |
| 002 140 | 006 | | LAI 012 | Load unsigned-variable |
| 002 141 | 012 | | | side of multiplier with |
| 002 142 | 131 | | OUT $\overline{\text{SEL12}}$ | 10 ($12_8$). |
| 002 143 | 115 | | INP $\overline{\text{SEL6}}$ | Input low order product |
| 002 144 | 370 | | LMA | into INP. REF. |
| 002 145 | 301 | | LAB | Load character into A. |
| 002 146 | 104 | | JMP INT | Go to INT. |
| 002 147 | 110 | | | |
| 002 150 | 002 | | | |
| 002 151 | 106 | FRC | CAL READ | Read character from TTY. |
| 002 152 | 050 | | | |
| 002 153 | 002 | | | |
| 002 154 | 301 | | LAB | Load Load character into A. |
| 003 155 | 074 | | CPI 015 | Is character a carriage |
| 002 156 | 015 | | | return? |
| 002 157 | 053 | | RTZ | If yes, return to call. |
| 002 160 | 024 | | SUI 060 | Convert ASCII to octal |
| 002 161 | 060 | | | number. |
| 002 062 | 204 | | ADE | Index table pointer. |
| 002 163 | 056 | | LHI 002 | Set memory page to 2. |
| 002 164 | 002 | | | |
| 002 165 | 360 | | LLA | Load memory pointer w/index. |
| 002 166 | 307 | | LAM | Load table value into A. |
| 002 167 | 056 | | LHI 003 | Set memory page to 3. |
| 002 170 | 003 | | | |
| 002 171 | 362 | | LLC | Load memory pointer w/alt. |
| 002 172 | 207 | | ADM | Add table value to fraction. |
| 002 173 | 370 | | LMA | Store result in fraction loc. |
| 002 174 | 006 | | LAI 012 | Move table pointer to |
| 002 175 | 012 | | | next table. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 002 176 | 204 | | ADE | |
| 002 177 | 340 | | LEA | |
| 002 200 | 104 | | JMP FRC | Go to FRC. |
| 002 201 | 151 | | | |
| 002 202 | 002 | | | |
| 002 203 | 307 | MUL | LAM | Load signed-variable side |
| 002 204 | 127 | | OUT SEL11 | of multiplier w/Coef.-Int. |
| 002 205 | 061 | | DCL | Decrement memory pointer |
| 002 206 | 307 | | LAM | Load unsigned variable side |
| 002 207 | 131 | | OUT SEL12 | of multiplier w/variable. |
| 002 210 | 113 | | INP SEL5 | Input high order of product. |
| 002 211 | 320 | | LCA | Load high order into C. |
| 002 212 | 115 | | INP SEL6 | Input low order of product. |
| 002 213 | 310 | | LBA | Load low order into B. |
| 002 214 | 061 | | DCL | Decrement memory pointer. |
| 002 215 | 307 | | LAM | Load add/subt. flag into A. |
| 002 216 | 074 | | CPI 000 | Is product to be added? |
| 002 217 | 000 | | | |
| 002 220 | 110 | | JFZ M3 | |
| 002 221 | 260 | | | If no, go to M3. |
| 002 222 | 002 | | | If yes, continue. |
| 002 123 | 303 | | LAD | Low order sum into A. |
| 002 224 | 201 | | ADB | Add low order product. |
| 002 225 | 330 | | LDA | Load sum into D. |
| 002 226 | 304 | | LAE | High order sum into A. |
| 002 227 | 212 | | ACC | Add high order product. |
| 002 230 | 340 | | LEA | Load sum into E. |
| 002 231 | 061 | | DCL | Decrement memory pointer. |
| 002 232 | 307 | | LAM | Load signed-variable side |
| 002 233 | 127 | | OUT SEL11 | of multiplier w/Coef.-frac. |
| 002 234 | 113 | | INP SEL5 | Input high order of product |
| 002 235 | 310 | | LBA | into Reg. B. |
| 002 236 | 026 | | LCI 000 | Clear pseudo-high order. |
| 002 237 | 000 | | | |
| 002 240 | 022 | | RAL | Rotate sign bit into carry. |
| 002 241 | 100 | | JFC M2 | Is product positive? |
| 002 242 | 246 | | | If yes, go to M2. |
| 002 243 | 002 | | | If no, continue. |
| 002 244 | 026 | | LCI 377 | Propogate sign bit |
| 002 245 | 377 | | | through pseudo-high order. |
| 002 246 | 032 | M2 | RAR | Rotate carry back. |
| 002 247 | 303 | | LAD | Low order sum into A. |
| 002 250 | 201 | | ADB | Add high order product. |
| 002 251 | 330 | | LDA | Sum into D. |
| 002 252 | 304 | | LAE | High order sum into A. |
| 002 253 | 212 | | ACC | Add pseudo-high order. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 002 254 | 340 | | LEA | Sum into E. |
| 002 255 | 104 | | JMP M4 | Go to M4. |
| 002 256 | 312 | | | |
| 002 257 | 002 | | | |
| 002 260 | 303 | M3 | LAD | Low order sum into A. |
| 002 261 | 221 | | SUB | Subtract low order product. |
| 002 262 | 330 | | LDA | Difference into D. |
| 002 263 | 304 | | LAE | High order sum into A. |
| 002 264 | 232 | | SBC | Subtract high order product. |
| 002 265 | 340 | | LEA | Difference into E. |
| 002 266 | 061 | | DCL | Decrement memory pointer. |
| 002 267 | 307 | | LAM | Load signed-variable side |
| 002 270 | 127 | | OUT SEL11 | of multiplier w/Coef. frac. |
| 002 271 | 113 | | INP SEL5 | Input high order product |
| 002 272 | 310 | | LBA | into Reg. B. |
| 002 273 | 026 | | LCI 000 | Clear pseudo-high order. |
| 002 274 | 000 | | | |
| 002 275 | 022 | | RAL | Rotate sign bit into carry. |
| 002 276 | 100 | | JFC M22 | Is product positive? |
| 002 277 | 303 | | | If yes, go to M22. |
| 002 300 | 002 | | | If no, continue. |
| 002 301 | 026 | | LCI 377 | Propogate sign bit |
| 002 302 | 377 | | | through pseudo-high order. |
| 002 303 | 032 | M22 | RAR | Rotate carry back. |
| 002 304 | 303 | | LAD | Low order sum into A. |
| 002 305 | 221 | | SUB | Subtract high order product. |
| 002 306 | 330 | | LDA | Difference into D. |
| 002 307 | 304 | | LAE | High order sum into A. |
| 002 310 | 232 | | SBC | Subtract pseudo-high order. |
| 002 311 | 340 | | LEA | Difference into E. |
| 002 312 | 061 | M4 | DCL | Decrement memory pointer. |
| 002 313 | 053 | | RTZ | Return, if zero, to call. |
| 002 314 | 104 | | JMP MUL | Go to MUL. |
| 002 315 | 203 | | | |
| 002 316 | 002 | | | |
| 002 322 | 320 | ALIN | LCA | Load FEEDBACK into C. |
| 002 323 | 301 | | LAB | Load INP. REF. into B. |
| 002 324 | 004 | | ADI 200 | Make INP. REF. compatible |
| 002 325 | 200 | | | with FEEDBACK. |
| 002 326 | 272 | | CPC | Is FEEDBACK greater than |
| 002 327 | 140 | | JTC AL1 | INP. REF.? |
| 002 330 | 341 | | | If yes, go to AL1. |
| 002 331 | 002 | | | If no, continue. |
| 002 332 | 222 | | SUC | Subtract FEEDBACK from |
| 002 333 | 074 | | CPI 200 | INP. REF. Does result exceed |
| 002 334 | 200 | | | upper limit? |
| 002 335 | 043 | | RTC | If no, return to call. |

| ADDRESS | OP-CODE | LABEL | INSTRUCTION | COMMENT |
|---|---|---|---|---|
| 002 336 | 006 | | LAI 177 | If yes, load upper limit |
| 002 337 | 177 | | | in place of result. |
| 002 340 | 007 | | RET | Return to call. |
| 002 341 | 310 | AL1 | LBA | Load INP. REF. into B. |
| 002 342 | 302 | | LAC | Load FEEDBACK into A. |
| 002 343 | 221 | | SUB | Subtract INP. REF. from |
| 002 344 | 074 | | CPI 177 | FEEDBACK. Does result |
| 002 345 | 177 | | | exceed upper limit? |
| 002 346 | 100 | | JFC AL2 | |
| 002 347 | 356 | | | If yes, go to AL2. |
| 002 350 | 002 | | | If no, continue. |
| 002 351 | 054 | | XRI 377 | Complement result. |
| 002 352 | 377 | | | |
| 002 353 | 004 | | ADI 001 | Add 1 to result. |
| 002 354 | 001 | | | |
| 002 355 | 007 | | RET | Return to call. |
| 002 356 | 006 | AL2 | LAI 201 | Load lower limit in place |
| 002 357 | 201 | | | of result. |
| 002 360 | 007 | | RET | Return to call. |

# VITA

Laurence Allan Phillips was born February 23, 1951, at Rapid City, South Dakota. He resided with his parents, Elwyn and Margaret, at Fort Meade, South Dakota, while attending elementary and junior high schools in Sturgis, South Dakota. At the age of fifteen he moved with his family to Johnson City, Tennessee, where he attended Science Hill High School.

In September, 1969, he enrolled in the pre-engineering curriculum at East Tennessee State University. In September, 1971, he transferred to the University of Tennessee and graduated in August, 1973, with a Bachelor of Science degree in Electrical Engineering. In September, 1973, he accepted a Teaching Assistantship at the University of Tennessee and enrolled in Graduate School to continue studies in Electrical Engineering.

He accepted summer employment with the Aluminum Company of America in June, 1974. After returning to the university to continue graduate studies in September, 1974, he accepted full time employment with the Aluminum Company of America in February, 1975.